SAS® 9.3 Formats and Informats

Reference

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2011. *SAS® 9.3 Formats and Informats: Reference*. Cary, NC: SAS Institute Inc.

SAS® 9.3 Formats and Informats: Reference

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

1st printing, July 2011
1st electronic book, November 2011
2nd printing, August 2012

Contents

About This Book

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

SAS uses standard conventions in the documentation of syntax for SAS language elements. These conventions enable you to easily identify the components of SAS syntax. The conventions can be divided into these parts:

- syntax components
- style conventions
- special characters
- references to SAS libraries and external files

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=).

keyword
: specifies the name of the SAS language element that you use when you write your program. Keyword is a literal that is usually the first word in the syntax. In a CALL routine, the first two words are keywords.

In the following examples of SAS syntax, the keywords are the first words in the syntax:

CHAR (*string, position*)

CALL RANBIN (*seed, n, p, x*);

ALTER (*alter-password*)

BEST *w*.

REMOVE <*data-set-name*>

In the following example, the first two words of the CALL routine are the keywords:

CALL RANBIN(*seed*, *n*, *p*, *x*)

The syntax of some SAS statements consists of a single keyword without arguments:

DO;
... *SAS code* ...

END;

Some system options require that one of two keyword values be specified:

DUPLEX | NODUPLEX

argument
: specifies a numeric or character constant, variable, or expression. Arguments follow the keyword or an equal sign after the keyword. The arguments are used by SAS to process the language element. Arguments can be required or optional. In the syntax, optional arguments are enclosed between angle brackets.

In the following example, *string* and *position* follow the keyword CHAR. These arguments are required arguments for the CHAR function:

CHAR (*string, position*)

Each argument has a value. In the following example of SAS code, the argument *string* has a value of 'summer', and the argument *position* has a value of 4:`x=char('summer', 4);`

In the following example, *string* and *substring* are required arguments, while *modifiers* and *startpos* are optional.

FIND(*string, substring* <,*modifiers*> <,*startpos*>

Note: In most cases, example code in SAS documentation is written in lowercase with a monospace font. You can use uppercase, lowercase, or mixed case in the code that you write.

Style Conventions

The style conventions that are used in documenting SAS syntax include uppercase bold, uppercase, and italic:

UPPERCASE BOLD
: identifies SAS keywords such as the names of functions or statements. In the following example, the keyword ERROR is written in uppercase bold:

ERROR<*message*>;

UPPERCASE
: identifies arguments that are literals.

In the following example of the CMPMODEL= system option, the literals include BOTH, CATALOG, and XML:

CMPMODEL = BOTH | CATALOG | XML

italics
: identifies arguments or values that you supply. Items in italics represent user-supplied values that are either one of the following:

- nonliteral arguments In the following example of the LINK statement, the argument *label* is a user-supplied value and is therefore written in italics:

  LINK *label*;

- nonliteral values that are assigned to an argument

  In the following example of the FORMAT statement, the argument DEFAULT is assigned the variable *default-format*:

  FORMAT = *variable-1* <, ..., *variable-nformat*><DEFAULT = *default-format*>;

Items in italics can also be the generic name for a list of arguments from which you can choose (for example, *attribute-list*). If more than one of an item in italics can be used, the items are expressed as *item-1, ..., item-n*.

Special Characters

The syntax of SAS language elements can contain the following special characters:

=

an equal sign identifies a value for a literal in some language elements such as system options.

In the following example of the MAPS system option, the equal sign sets the value of MAPS:

MAPS = *location-of-maps*

< >

angle brackets identify optional arguments. Any argument that is not enclosed in angle brackets is required.

In the following example of the CAT function, at least one item is required:

CAT (*item-1* <, ..., *item-n*>)

|

a vertical bar indicates that you can choose one value from a group of values. Values that are separated by the vertical bar are mutually exclusive.

In the following example of the CMPMODEL= system option, you can choose only one of the arguments:

CMPMODEL = BOTH | CATALOG | XML

...

an ellipsis indicates that the argument or group of arguments following the ellipsis can be repeated. If the ellipsis and the following argument are enclosed in angle brackets, then the argument is optional.

In the following example of the CAT function, the ellipsis indicates that you can have multiple optional items:

CAT (*item-1* <, ..., *item-n*>)

'*value*' or “*value*”

indicates that an argument enclosed in single or double quotation marks must have a value that is also enclosed in single or double quotation marks.

In the following example of the FOOTNOTE statement, the argument *text* is enclosed in quotation marks:

FOOTNOTE <*n*> <*ods-format-options* '*text*' | “*text*”>;

;

a semicolon indicates the end of a statement or CALL routine.

In the following example each statement ends with a semicolon: `data namegame; length color name $8; color = 'black'; name = 'jack'; game = trim(color) || name; run;`

References to SAS Libraries and External Files

Many SAS statements and other language elements refer to SAS libraries and external files. You can choose whether to make the reference through a logical name (a libref or fileref) or use the physical filename enclosed in quotation marks. If you use a logical name, you usually have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the association. Several methods of referring to SAS libraries and external files are available, and some of these methods depend on your operating environment.

In the examples that use external files, SAS documentation uses the italicized phrase *file-specification*. In the examples that use SAS libraries, SAS documentation uses the italicized phrase *SAS-library*. Note that *SAS-library* is enclosed in quotation marks:

```
infile file-specification obs = 100;
libname libref 'SAS-library';
```

What's New in SAS 9.3 Formats and Informats

Overview

The SAS formats and informats are now published as a separate document. They are no longer part of *SAS Language Reference: Dictionary*. For more information, see "Changes to SAS Language Reference: Dictionary" on page ix.

For SAS 9.3, there are no new or enhanced formats.

New informats read IBM date and time values that include a century marker, read Java date and time values, and read hours, minutes, and seconds in the form *hhmmss* or *hh:mm:ss*.

New SAS Informats

The following informats are new:

B8601CI*w*. (p. 250)
: reads an IBM date and time value that includes a century marker, in the form *cyymmddhhmmss<fff>*.

B8601DJ*w*. (p. 253)
: reads a Java date and time value that is in the form *yyyymmddhhmmssffffff*.

HHMMSS*w*. (p. 284)
: reads hours, minutes, and seconds in the form *hhmmss* or *hh*:*mm*:*ss*.

Changes to *SAS Language Reference: Dictionary*

Prior to SAS 9.3, this document was part of *SAS Language Reference: Dictionary*. Starting with SAS 9.3, *SAS Language Reference: Dictionary* has been divided into seven documents:

- *SAS Data Set Options: Reference*
- *SAS Formats and Informats: Reference*
- *SAS Functions and CALL Routines: Reference*
- *SAS Statements: Reference*

- *SAS System Options: Reference*
- *SAS Component Objects: Reference* (contains the documentation for the Hash Object and the Java Object)
- *Base SAS Utilities: Reference* (contains the documentation for the SAS DATA step debugger and the SAS Utility macro %DS2CSV)

Recommended Reading

Here is the recommended reading list for this title:

- *An Array of Challanges-Test Your SAS Skills*
- **Base SAS Glossary**
- *Base SAS Procedures Guide*
- *Debugging SAS Programs: A Handbook of Tools and Techniques*
- *The Little SAS Book: A Primer*
- *SAS Companion for UNIX Environments*
- *SAS Companion for Windows*
- *SAS Companion for z/OS*
- *SAS Guide to Report Writing: Examples*
- *SAS Language Reference: Concepts*
- *SAS National Language Support (NLS): Reference Guide*
- *SAS Programming by Example*
- *The SAS Workbook*
- *Step-by-Step Programming with Base SAS Software*
- *Using the SAS Windowing Environment: A Quick Tutorial*

For a complete list of SAS publications, go to support.sas.com/bookstore. If you have questions about which titles you need, please contact a SAS Publishing Sales Representative:

SAS Publishing Sales
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-3228
Fax: 1-919-677-8166
E-mail: sasbook@sas.com
Web address: support.sas.com/bookstore

Part 1

SAS Formats

Chapter 1

About Formats

Definition of Formats

A format is a type of SAS language element that applies a pattern to or executes instructions for a data value to be displayed or written as output. Types of formats correspond to the data's type: numeric, character, date, time, or timestamp. The ability to create user-defined formats is also supported. Examples of SAS formats are BINARY, DATE, and WORDS. 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.

  See: For information about user-defined formats, see Chapter 23, “FORMAT Procedure” in *Base 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. 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. The remaining columns include 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 attempt fails, an error message that describes the problem appears in the SAS log.

When the value of *d* is greater than fifteen, the precision of the decimal value after the 15th significant digit might not be accurate.

Using Formats

Ways to Specify Formats

About Specifying 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*w.d* format to write the numeric value for AMOUNT as a dollar amount:

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

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

```
$1,145.32
```

For more information, see “PUT Statement” in *SAS Statements: Reference*.

PUT Function

The PUT function converts a numeric variable, a character variable, or a constant using 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 returns a value of 0F, which is assigned to the variable CHAR.

The PUT function is useful for converting a numeric value to a character value.

For more information, see “PUT Function” in *SAS Functions and CALL Routines: Reference*.

%SYSFUNC Macro Function

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 “%SYSFUNC and %QSYSFUNC Functions” in *SAS Macro Language: Reference*.

FORMAT Statement

The FORMAT statement permanently associates character variables with character formats and numeric variables with numeric formats.

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

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

Because the FORMAT statement permanently associates a format with a variable, any subsequent DATA step or PROC step uses COMMA10.2 to write the values of SALES1, SALES2, and SALES3.

For more information, see “FORMAT Statement” in *SAS Statements: Reference*.

Note: If you assign formats with a FORMAT statement before a PUT statement, all leading blanks are trimmed. Formats that are associated with variables by using a FORMAT statement behave like formats that are used with a colon (:) modifier in a subsequent PUT statement. For details about using the colon format modifier, see “PUT Statement, List” in *SAS Statements: Reference*.

ATTRIB Statement

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

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

Because the ATTRIB statement permanently associates a format with a variable, any subsequent DATA step or PROC step uses COMMA10.2 to write the values of SALES1, SALES2, and SALES3.

For more information, see “ATTRIB Statement” in *SAS Statements: Reference*.

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 about using formats in SAS procedures, see “Formatted Values” in Chapter 2 of *Base 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 enables you to create your own formats for both character and numeric variables.

For more information, see Chapter 23, "FORMAT Procedure" in *Base 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 Chapter 23, "FORMAT Procedure" in *Base 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	Result
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 for Integer Binary Data on Big Endian and Little Endian Platforms

Definitions

Integer values for binary integer data 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 on SPARC. The following platforms are considered little endian: Intel ABI, Linux, OpenVMS Alpha, OpenVMS Integrity, Solaris on x64, Tru64 UNIX, 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. These representations result from the output of the integer binary value –2 expressed in hexadecimal representation.

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 Signed Integer column, “no” indicates that the number is unsigned and cannot be negative. “Yes” indicates that the number can be either negative or positive.

Table 1.1 SAS Formats and Byte Ordering

Platform For Which the Data Was Created	Platform That Writes the Data	Signed Integer	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.

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

Data Conversions and Encodings

An encoding maps each character in a character set to a unique numeric representation, resulting in a table of all code points. A single character can have different numeric representations in different encodings. For example, the ASCII encoding for the dollar symbol $ is 24 hexadecimal. The Danish EBCDIC encoding for the dollar symbol $ is 67 hexadecimal. In order for a version of SAS that normally uses ASCII to properly interpret a data set that is encoded in Danish EBCDIC, the data must be transcoded.

Transcoding is the process of moving data from one encoding to another. When transcoding the ASCII dollar sign to the Danish EBCDIC dollar sign, the hexadecimal representation for the character is converted from the value 24 to a 67.

If you want to know the encoding of a particular SAS data set, for SAS 9 and above follow these steps:

1. Locate the data set with SAS Explorer.
2. Right-click the data set.
3. Select Properties from the menu.
4. Click the Details tab.
5. The encoding of the data set is listed, along with other information.

Some situations where data might commonly be transcoded are:

- when you share data between two different SAS sessions that are running in different locales or in different operating environments,

- when you perform text-string operations, such as converting to uppercase or lowercase,
- when you display or print characters from another language,
- when you copy and paste data between SAS sessions running in different locales.

For more information about SAS features designed to handle Transcoding for NLS from different encodings or operating environments, see *SAS National Language Support (NLS): Reference Guide*.

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 might 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 character. For example, the value 15 is stored in two nibbles, using the hexadecimal characters 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 character. 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= system option 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 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 and Informats
PIC S9(X) PACKED-DECIMAL	S370FPDw.

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

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

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

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

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

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

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

Summary of Packed Decimal and Zoned Decimal Formats and Informats

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

Format	Type of data representation	Corresponding informat	Comments
PD	Packed decimal	PD	Local signed packed decimal

Format	Type of data representation	Corresponding informat	Comments
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 (hexadecimal 40) to EBCDIC zero (hexadecimal F0); corresponds to the informat as zoned decimal
none	Zoned decimal	ZDV	Non-IBM zoned decimal representation
S370FPD	Packed decimal	S370FPD	Last nibble C (positive) or D (negative)
S370FPDU	Packed decimal	S370FPDU	Last nibble always F (positive)
S370FZD	Zoned decimal	S370FZD	Last byte contains sign in upper nibble: C (positive) or D (negative)
S370FZDU	Zoned decimal	S370FZDU	Unsigned; sign nibble always F
S370FZDL	Zoned decimal	S370FZDL	Sign nibble in first byte in informat; separate leading sign byte of hexadecimal C0 (positive) or D0 (negative) in format
S370FZDS	Zoned decimal	S370FZDS	Leading sign of - (hexadecimal 60) or + (hexadecimal 4E)
S370FZDT	Zoned decimal	S370FZDT	Trailing sign of - (hexadecimal 60) or + (hexadecimal 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

Format	Type of data representation	Corresponding informat	Comments
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

Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations

ISO 8601 Formatting Symbols

The following list explains the formatting symbols that are used to notate the ISO 8601 dates, time, datetime, durations, and interval values:

n
: specifies a number that represents the number of years, months, or days

P
: indicates that the duration that follows is specified by the number of years, months, days, hours, minutes, and seconds

T
: indicates that a time value follows. Any value with a time must begin with T.

  Requirement: Time values that are read by the extended notation informats that begin with the characters E8601 must use an uppercase T.

W
: indicates that the duration is specified in weeks.

Z
: indicates that the time value is the time in Greenwich, England, or UTC time.

+|-
: the + indicates the time zone offset to the east of Greenwich, England. The - indicates the time zone offset to the west of Greenwich, England.

yyyy
specifies a four-digit year

mm
as part of a date, specifies a two-digit month, 01–12

dd
specifies a two-digit day, 01–1

hh
specifies a two-digit hour, 00–24

mm
as part of a time, specifies a two-digit minute, 00–59

ss
specifies a two-digit second, 00–59

fff | ffffff
specifies an optional fraction of a second using the digits 0–9:

fff	use 1 - 3 digits for values read by the $N8601B informat and the $N8601E informat
ffffff	use 1 - 6 digits for informat other than the $N8601B informat and the $N8601E informat

Y
indicates that a year value proceeds this character in a duration

M
as part of a date, indicates that a month value proceeds this character in a duration

D
indicates that a day value proceeds this character in a duration

H
indicates that an hour value proceeds this character in a duration

M
as part of a time, indicates that a minute value proceeds this character in a duration

S
indicates that a seconds value proceeds this character in a duration

Writing ISO 8601 Date, Time, and Datetime Values

SAS uses the formats in the following table to write date, time, and datetime values in the ISO 8601 basic and extended notations from SAS date, time, and datetime values.

Date, Time, or Datetime	ISO 8601 Notation	Example	Format
Basic Notations			
Date	*yyyymmdd*	20120915	B8601DA*w.*
Time	*hhmmssffffff*	155300322348	B8601TM*w.d*
Time with time zone	*hhmmss+|-hhmm*	155300+0500	B8601TZ*w.d*

Date, Time, or Datetime	ISO 8601 Notation	Example	Format
	*hhmmss*Z	155300Z	B8601TZ*w.d*
Convert to local time with time zone	hhmmss+|-hhmm	155300+0500	B8601LZ*w.d*
Datetime	*yyyymmdd*T*hhmmssffffff*	20120915T155300	B8601DT*w.d*
Datetime with timezone	*yyyymmdd*T*hhmmss*+|-*hhmm*	20120915T155300+0500	B8601DZ*w.d*
	*yyyymmdd*T*hhmmss*Z	20120915T155300Z	B8601DZ*w.d*
Write the date from a datetime	*yyyymmdd*	20120915	B8601DN*w*.
Extended Notations			
Date	*yyyy-mm-dd*	2012-09-15	E8601DA*w*.
Time	*hh:mm:ss.ffffff*	15:53:00.322348	E8601TM*w.d*
Time with time zone	*hh:mm:ss.fffff*f+|-*hh:mm*	15:53:00+05:00	E8601TZ*w.d*
Convert to local time with time zone	*hh:mm:ss.ffffff*+|-*hh:mm*	15:53:00+05:00	E8601LZ*w.d*
Datetime	*yyyy-mm-dd*T*hh:mm:ss.ffffff*	2012-09-15T15:53:00	E8601DT*w.d*
Datetime with time zone	*yyyy-mm-dd*T*hh:mm:ss.nnnnnn*+|-*hh:mm*	2012-09-15T15:53:00+05:00	E8601DZ*w.d*
Write the date from a datetime	*yyyy-mm-dd*	2012-09-15	E8601DN*w*.

An asterisk (*) used in place of a date or time formatted value that is out-of-range.

Writing ISO 8601 Duration, Datetime, and Interval Values

Duration, Datetime, and Interval Formats

SAS writes duration, datetime, and interval values from character data using these formats:

Table 1.2 *Complete Component Forms*

Time Component	ISO 8601 Notation	Example	Format
Duration - Basic Notation	P*yyyymmdd*T*hhmmssfff*	P20120915T155300	$N8601BA
	-P*yyyymmdd*T*hhmmssfff*	-P20120915T155300	$N8601BA

Time Component	ISO 8601 Notation	Example	Format
Duration - Extended Notation	P*yyyy-mm-dd*T*hh:mm:ss.fff*	P2012-09-15T15:53:00	$N8601EA
	-P*yyyy-mm-dd*T*hh:mm:ss.fff*	-P2012-09-15T15:53:00	$N8601EA
Duration - Basic and Extended Notation	P*n*Y*n*M*n*DT*n*H*n*M*n*S	P2y10m14dT20h13m45s	$N8601B $N8601E
	-P*n*Y*n*M*n*DT*n*H*n*M*n*S	-P2y10m14dT20h13m45s	$N8601B $N8601E
	P*n*W (weeks)	P6w	$N8601B $N8601E
Interval - Basic Notation	*yyyymmdd*T*hhmmssfff*/ yyyymmddT*hhmmssfff*	20120915T155300/20141113T000000	$N8601BA
	P*n*Y*n*M*n*DT*n*H*n*M*n*S/ *yyyymmdd*T*hhmmssfff*	P2y10M14dT20h13m45s/ 20120915T155300	$N8601B
	*yyyymmdd*T*hhmmssfff*/ P*n*Y*n*M*n*DT*n*H*n*M*n*S	20120915T155300/ P2y10M14dT20h13m45s	$N8601BA
Interval- Extended Notation	*yyyy-mm-dd*T*hh:mm:ss.fff*/ *yyyy-mm-dd*T*hh:mm:ss.fff*	2012-09-15T15:53:00/2014-11-13T00:00:00	$N8601EA
	P*n*Y*n*M*n*DT*n*H*n*M*n*S/*yyyy-mm-dd*T*hh:mm:ss.fff*	P2y10M14dT20h13m45s/ 2012-09-15T15:53:00	$N8601E
	*yyyy-mm-dd*T*hh:mm:ss.fff*/ P*n*Y*n*M*n*DT*n*H*n*M*n*S	2012-09-15T15:53:00/ P2y10M14dT20h13m45s	$N8601EA
Datetime-Basic Notation	*yyyymmdd*T*hhmmss.fff*+|-*hhmm*	20120915T155300	$N8601BA
	(all blank)		$N8601B $N8601BA $N8601E $N8601EA
Datetime-Extended Notation	*yyyy-mm-dd*T*hh:mm:ss.fff* +|-*hhmm*	2012-09-15T15:53:00 +04:30	$N8601EA
	(all blank)		$N8601B $N8601BA $N8601E $N8601EA

Writing Omitted Components

An omitted component can be represented by a hyphen (-) or an x in the extended datetime form *yyyy-mm-dd*T*hh:mm:ss* and in the extended duration form P*yyyy-mm-dd*T*hh:mm:ss*.

Omitted components in the durations form P*n*Y*n*M*n*DT*n*H*n*M*n*S are dropped, they do not contain a hyphen or x. For example, P2mT4H.

The following formats write omitted components that use the hyphen and the x:

Format	Datetime Form	Duration Form	Examples
$N8601H	*yyyy-mm-dd*T*hh:mm:ss*	P*n*Y*n*M*n*DT*n*H*n*M*n*S	--09-15T15:-:53 P2Y2DT4H5M6S/--09-15T15:-:00
$N8601EH	*yyyy-mm-dd*T*hh:mm:ss*	P*yyyy-mm-dd*T*hh:mm:ss*	P000---02T02:55:20/ 2012---15T-:-:45
$N8601X	*yyyy-mm-dd*T*hh:mm:ss*	P*n*Y*n*M*n*DT*n*H*n*M*n*S	P2Y2DT4H5M6S/ x-09-15T15:x:00
$N8601EX	*yyyy-mm-dd*T*hh:mm:ss*	P*yyyy-mm-dd*T*hh:mm:ss*	P0003-x-02T02:55:20/2012-x-15Tx:x:45

Datetime values with omitted components that are formatted with either the $N8601B format or the $N8601BA format are formatted in the extended notation using the hyphen for omitted components to ensure accurate data. For example, when the month is an omitted component, the value 2012---15 is written and not 2012-15.

The extended notation with hyphens is also used in place of the basic notation if a duration is formatted by using the $N8601BA format. Using the same date, P2012---15 is written and not P2012-15.

Writing Truncated Duration, Datetime, and Interval Values

Duration, datetime, or interval values can be truncated when one or more lower order values is 0 or is not significant. When SAS writes a truncated value using the formats $N8601B, $N8601BA, $N8601E, and $N8601EA, the display of the value stops at the last nonmissing component.

When you format a truncated value by using either the $N8601H format or the $N8601EH format, the lower order components are written with a hyphen. When you format a truncated value by using the $N8601X format or the $N8601EX format, the lower order components are written with an x.

The following examples show truncated values:

- `p00030202T1031`
- `2012-09-15T15/2014-09-15T15:53`
- `-p0003-03-03T-:-:-`
- `P2y3m4dT5h6m`
- `2012-09-xTx:x:x`
- `2012`

Normalizing Duration Components

When a value for a duration component is greater than the largest standard value for a component, SAS normalizes the component except when the duration component is a single component. The following table shows examples of normalized duration components:

Duration	Extended Normalized Duration
`p3y13m`	`p0004-01`
`pt24h24m65s`	`P----01T-:25:05`
`p3y13mT24h61m`	`P0004-01-01T01:01`
`p0004-13`	`p0005-01`
`p0003-02-61T15:61:61`	`P0003-04-01T16:02:01`
`p13m`	`P13M`

If a component contains the largest value, such as 60 for minutes or seconds, SAS normalizes the value and replaces the value with a hyphen. For example, **`pT12:60:13`** becomes **`PT13:-:13`**.

Thirty days is used to normalize a month.

Dates and times in a datetime value that are greater than the standard value for the component are not normalized. They produce an error.

Fractions in Durations, Datetime, and Interval Values

Ending components can contain a fraction that consists of a period or a comma, followed by one to three digits. The following examples show the use of fractions in duration, datetime, and interval values:

- **`201209.5`**
- **`P2012-09-15T10.33`**
- **`2012-09-15/P0003-03-03,333`**

Chapter 2
Dictionary of Formats

Formats Documented in Other Publications

For informats that support national language, see Chapter 9, “Format Entries,” in *SAS National Language Support (NLS): Reference Guide*.

Formats by Category

There are four categories of formats in this list:

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.
ISO 8601	instructs SAS to write date, time, and datetime values using the ISO 8601 standard.
Numeric	instructs SAS to write numeric data values from numeric variables.

Formats that support national languages can be found in *SAS National Language Support (NLS): Reference Guide*.

Storing user-defined formats is an important consideration if you associate these formats with variables in permanent SAS data sets, especially those data sets shared with other users.For information about creating and storing user-defined formats, see Chapter 23, “FORMAT Procedure” in *Base SAS Procedures Guide*.

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

Category	Language Elements	Description
Character	$ASCIIw. Format (p. 33)	Converts native format character data to ASCII representation.
	$BASE64Xw. Format (p. 34)	Converts character data into ASCII text by using Base 64 encoding.
	$BINARYw. Format (p. 35)	Converts character data to binary representation.
	$CHARw. Format (p. 36)	Writes standard character data.
	$EBCDICw. Format (p. 37)	Converts native format character data to EBCDIC representation.
	$HEXw. Format (p. 38)	Converts character data to hexadecimal representation.
	$MSGCASEw. Format (p. 39)	Writes character data in uppercase when the MSGCASE system option is in effect.
	$OCTALw. Format (p. 49)	Converts character data to octal representation.
	$QUOTEw. Format (p. 50)	Writes data values that are enclosed in double quotation marks.
	$REVERJw. Format (p. 52)	Writes character data in reverse order and preserves blanks.
	$REVERSw. Format (p. 52)	Writes character data in reverse order and left aligns
	$UPCASEw. Format (p. 53)	Converts character data to uppercase.
	$VARYINGw. Format (p. 54)	Writes character data of varying length.
	$w. Format (p. 56)	Writes standard character data.

Category	Language Elements	Description
Date and Time	$N8601Bw.d Format (p. 39)	Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyymmddThhmmss.
	$N8601BAw.d Format (p. 41)	Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PyyyymmddThhmmss and yyyymmddThhmmss.
	$N8601Ew.d Format (p. 42)	Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss.
	$N8601EAw.d Format (p. 43)	Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh:mm:ss.
	$N8601EHw.d Format (p. 44)	Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh:mm:ss, using a hyphen (-) for omitted components.
	$N8601EXw.d Format (p. 46)	Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh:mm:ss, using an x for each digit of an omitted component.
	$N8601Hw.d Format (p. 47)	Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss, dropping omitted components in duration values and using a hyphen (-) for omitted components in datetime values.
	$N8601Xw.d Format (p. 48)	Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss, dropping omitted components in duration values and using an x for each digit of an omitted component in datetime values.
	B8601DAw. Format (p. 60)	Writes date values by using the ISO 8601 basic notation yyyymmdd.
	B8601DNw. Format (p. 61)	Writes dates from datetime values by using the ISO 8601 basic notation yyyymmdd.
	B8601DTw.d Format (p. 62)	Writes datetime values by using the ISO 8601 basic notation yyyymmddThhmmss<ffffff>.
	B8601DZw. Format (p. 64)	Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+0000.
	B8601LZw. Format (p. 65)	Writes time values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic time notation hhmmss+|–hhmm.
	B8601TMw.d Format (p. 66)	Writes time values by using the ISO 8601 basic notation hhmmss<ffff>.

Category	Language Elements	Description
	B8601TZw. Format (p. 67)	Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 basic time notation hhmmss+|–hhmm.
	DATEw. Format (p. 73)	Writes date values in the form ddmmmyy, ddmmmyyyy, or dd-mmm-yyyy.
	DATEAMPMw.d Format (p. 74)	Writes datetime values in the form ddmmmyy:hh:mm:ss.ss with AM or PM.
	DATETIMEw.d Format (p. 75)	Writes datetime values in the form ddmmmyy:hh:mm:ss.ss.
	DAYw. Format (p. 77)	Writes date values as the day of the month.
	DDMMYYw. Format (p. 78)	Writes date values in the form ddmm<yy>yy or dd/mm/<yy>yy, where a forward slash is the separator and the year appears as either 2 or 4 digits.
	DDMMYYxw. Format (p. 79)	Writes date values in the form ddmm<yy>yy or dd-mm-yy<yy>, where the x in the format name is a character that represents the special character that separates the day, month, and year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.
	DOWNAMEw. Format (p. 84)	Writes date values as the name of the day of the week.
	DTDATEw. Format (p. 84)	Expects a datetime value as input and writes date values in the form ddmmmyy or ddmmmyyyy.
	DTMONYYw. Format (p. 86)	Writes the date part of a datetime value as the month and year in the form mmmyy or mmmyyyy.
	DTWKDATXw. Format (p. 87)	Writes the date part of a datetime value as the day of the week and the date in the form day-of-week, dd month-name yy (or yyyy).
	DTYEARw. Format (p. 88)	Writes the date part of a datetime value as the year in the form yy or yyyy.
	DTYYQCw. Format (p. 89)	Writes the date part of a datetime value as the year and the quarter and separates them with a colon (:).
	E8601DAw. Format (p. 91)	Writes date values by using the ISO 8601 extended notation yyyy-mm-dd.
	E8601DNw. Format (p. 92)	Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd.
	E8601DTw.d Format (p. 93)	Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.ffffff.

Category	Language Elements	Description
	E8601DZw. Format (p. 94)	Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00.
	E8601LZw. Format (p. 96)	Writes time values as local time, appending the Coordinated Universal Time (UTC) offset for the local SAS session, using the ISO 8601 extended time notation hh:mm:ss+|–hh:mm.
	E8601TMw.d Format (p. 97)	Writes time values by using the ISO 8601 extended notation hh:mm:ss.ffffff.
	E8601TZw.d Format (p. 98)	Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation hh:mm:ss+|–hh:mm.
	HHMMw.d Format (p. 103)	Writes time values as hours and minutes in the form hh:mm.
	HOURw.d Format (p. 105)	Writes time values as hours and decimal fractions of hours.
	JULDAYw. Format (p. 110)	Writes date values as the Julian day of the year.
	JULIANw. Format (p. 111)	Writes date values as Julian dates in the form yyddd or yyyyddd.
	MDYAMPMw.d Format (p. 112)	Writes datetime values in the form mm/dd/yy<yy> hh:mm AM|PM. The year can be either two or four digits.
	MMDDYYw. Format (p. 113)	Writes date values in the form mmdd<yy>yy or mm/dd/<yy>yy, where a forward slash is the separator and the year appears as either 2 or 4 digits.
	MMDDYYxw. Format (p. 115)	Writes date values in the form mmdd<yy>yy or mm-dd-<yy>yy, where the x in the format name is a character that represents the special character which separates the month, day, and year. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.
	MMSSw.d Format (p. 117)	Writes time values as the number of minutes and seconds since midnight.
	MMYYw. Format (p. 118)	Writes date values in the form mmM<yy>yy, where M is the separator and the year appears as either 2 or 4 digits.
	MMYYxw. Format (p. 119)	Writes date values in the form mm<yy>yy or mm-<yy>yy, where the x in the format name is a character that represents the special character that separates the month and the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.
	MONNAMEw. Format (p. 121)	Writes date values as the name of the month.
	MONTHw. Format (p. 122)	Writes date values as the month of the year.

Category	Language Elements	Description
	MONYYw. Format (p. 123)	Writes date values as the month and the year in the form mmmyy or mmmyyyy.
	PDJULGw. Format (p. 128)	Writes packed Julian date values in the hexadecimal format yyyydddF for IBM.
	PDJULIw. Format (p. 130)	Writes packed Julian date values in the hexadecimal format ccyydddF for IBM.
	QTRw. Format (p. 138)	Writes date values as the quarter of the year.
	QTRRw. Format (p. 139)	Writes date values as the quarter of the year in Roman numerals.
	TIMEw.d Format (p. 157)	Writes time values as hours, minutes, and seconds in the form hh:mm:ss.ss.
	TIMEAMPMw.d Format (p. 158)	Writes time and datetime values as hours, minutes, and seconds in the form hh:mm:ss.ss with AM or PM.
	TODw.d Format (p. 160)	Writes SAS time values and the time portion of SAS datetime values in the form hh:mm:ss.ss.
	WEEKDATEw. Format (p. 165)	Writes date values as the day of the week and the date in the form day-of-week, month-name dd, yy (or yyyy).
	WEEKDATXw. Format (p. 167)	Writes date values as the day of the week and date in the form day-of-week, dd month-name yy (or yyyy).
	WEEKDAYw. Format (p. 168)	Writes date values as the day of the week.
	WEEKUw. Format (p. 169)	Writes a week number in decimal format by using the U algorithm.
	WEEKVw. Format (p. 171)	Writes a week number in decimal format by using the V algorithm.
	WEEKWw. Format (p. 173)	Writes a week number in decimal format by using the W algorithm.
	WORDDATEw. Format (p. 175)	Writes date values as the name of the month, the day, and the year in the form month-name dd, yyyy.
	WORDDATXw. Format (p. 176)	Writes date values as the day, the name of the month, and the year in the form dd month-name yyyy.
	YEARw. Format (p. 179)	Writes date values as the year.
	YYMMw. Format (p. 180)	Writes date values in the form <yy>yyMmm, where M is a character separator to indicate that the month number follows the M and the year appears as either 2 or 4 digits.
	YYMMDDw. Format (p. 181)	Writes date values in the form yymmdd or <yy>yy-mm-dd, where a hyphen is the separator and the year appears as either 2 or 4 digits.

Category	Language Elements	Description
	YYMMDDxw. Format (p. 183)	Writes date values in the form yymmdd or <yy>yy-mm-dd, where the x in the format name is a character that represents the special character which separates the year, month, and day. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.
	YYMONw. Format (p. 186)	Writes date values in the form yymmm or yyyymmm.
	YYQw. Format (p. 187)	Writes date values in the form <yy>yyQq, where Q is the separator, the year appears as either 2 or 4 digits, and q is the quarter of the year.
	YYQxw. Format (p. 188)	Writes date values in the form <yy>yyq or <yy>yy-q, where the x in the format name is a character that represents the special character that separates the year and the quarter or the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.
	YYQRw. Format (p. 190)	Writes date values in the form <yy>yyQqr, where Q is the separator, the year appears as either 2 or 4 digits, and qr is the quarter of the year expressed in roman numerals.
	YYQRxw. Format (p. 191)	Writes date values in the form <yy>yyqr or <yy>yy-qr, where the x in the format name is a character that represents the special character that separates the year and the quarter or the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits and qr is the quarter of the year expressed in roman numerals.
ISO 8601	$N8601Bw.d Format (p. 39)	Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyymmddThhmmss.
	$N8601BAw.d Format (p. 41)	Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PyyyymmddThhmmss and yyyymmddThhmmss.
	$N8601Ew.d Format (p. 42)	Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss.
	$N8601EAw.d Format (p. 43)	Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh:mm:ss.
	$N8601EHw.d Format (p. 44)	Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh:mm:ss, using a hyphen (-) for omitted components.
	$N8601EXw.d Format (p. 46)	Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh:mm:ss, using an x for each digit of an omitted component.
	$N8601Hw.d Format (p. 47)	Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss, dropping

Category	Language Elements	Description
		omitted components in duration values and using a hyphen (-) for omitted components in datetime values.
	$N8601Xw.d Format (p. 48)	Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss, dropping omitted components in duration values and using an x for each digit of an omitted component in datetime values.
	B8601DAw. Format (p. 60)	Writes date values by using the ISO 8601 basic notation yyyymmdd.
	B8601DNw. Format (p. 61)	Writes dates from datetime values by using the ISO 8601 basic notation yyyymmdd.
	B8601DTw.d Format (p. 62)	Writes datetime values by using the ISO 8601 basic notation yyyymmddThhmmss<ffffff>.
	B8601DZw. Format (p. 64)	Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+0000.
	B8601LZw. Format (p. 65)	Writes time values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic time notation hhmmss+|–hhmm.
	B8601TMw.d Format (p. 66)	Writes time values by using the ISO 8601 basic notation hhmmss<ffff>.
	B8601TZw. Format (p. 67)	Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 basic time notation hhmmss+|–hhmm.
	E8601DAw. Format (p. 91)	Writes date values by using the ISO 8601 extended notation yyyy-mm-dd.
	E8601DNw. Format (p. 92)	Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd.
	E8601DTw.d Format (p. 93)	Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.ffffff.
	E8601DZw. Format (p. 94)	Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00.
	E8601LZw. Format (p. 96)	Writes time values as local time, appending the Coordinated Universal Time (UTC) offset for the local SAS session, using the ISO 8601 extended time notation hh:mm:ss+|–hh:mm.
	E8601TMw.d Format (p. 97)	Writes time values by using the ISO 8601 extended notation hh:mm:ss.ffffff.
	E8601TZw.d Format (p. 98)	Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation hh:mm:ss+|–hh:mm.

Category	Language Elements	Description
Numeric	BESTw. Format (p. 57)	SAS chooses the best notation.
	BESTDw.p Format (p. 58)	Prints numeric values, lining up decimal places for values of similar magnitude, and prints integers without decimals.
	BINARYw. Format (p. 60)	Converts numeric values to binary representation.
	COMMAw.d Format (p. 69)	Writes numeric values with a comma that separates every three digits and a period that separates the decimal fraction.
	COMMAXw.d Format (p. 70)	Writes numeric values with a period that separates every three digits and a comma that separates the decimal fraction.
	Dw.p Format (p. 71)	Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude.
	DOLLARw.d Format (p. 81)	Writes numeric values with a leading dollar sign, a comma that separates every three digits, and a period that separates the decimal fraction.
	DOLLARXw.d Format (p. 82)	Writes numeric values with a leading dollar sign, a period that separates every three digits, and a comma that separates the decimal fraction.
	Ew. Format (p. 90)	Writes numeric values in scientific notation.
	FLOATw.d Format (p. 100)	Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the dth power.
	FRACTw. Format (p. 101)	Converts numeric values to fractions.
	HEXw. Format (p. 102)	Converts real binary (floating-point) values to hexadecimal representation.
	IBw.d Format (p. 106)	Writes native integer binary (fixed-point) values, including negative values.
	IBRw.d Format (p. 108)	Writes integer binary (fixed-point) values in Intel and DEC formats.
	IEEEw.d Format (p. 109)	Generates an IEEE floating-point value by multiplying a number by 10 raised to the dth power.
	NEGPARENw.d Format (p. 124)	Writes negative numeric values in parentheses.
	NUMXw.d Format (p. 125)	Writes numeric values with a comma in place of the decimal point.
	OCTALw. Format (p. 126)	Converts numeric values to octal representation.
	PDw.d Format (p. 127)	Writes data in packed decimal format.

Category	Language Elements	Description
	PERCENTw.d Format (p. 131)	Writes numeric values as percentages.
	PERCENTNw.d Format (p. 132)	Produces percentages, using a minus sign for negative values.
	PIBw.d Format (p. 133)	Writes positive integer binary (fixed-point) values.
	PIBRw.d Format (p. 135)	Writes positive integer binary (fixed-point) values in Intel and DEC formats.
	PKw.d Format (p. 136)	Writes data in unsigned packed decimal format.
	PVALUEw.d Format (p. 137)	Writes p-values.
	RBw.d Format (p. 140)	Writes real binary data (floating-point) in real binary format.
	ROMANw. Format (p. 141)	Writes numeric values as roman numerals.
	S370FFw.d Format (p. 142)	Writes native standard numeric data in IBM mainframe format.
	S370FIBw.d Format (p. 143)	Writes integer binary (fixed-point) values, including negative values, in IBM mainframe format.
	S370FIBUw.d Format (p. 144)	Writes unsigned integer binary (fixed-point) values in IBM mainframe format.
	S370FPDw.d Format (p. 146)	Writes packed decimal data in IBM mainframe format.
	S370FPDUw.d Format (p. 147)	Writes unsigned packed decimal data in IBM mainframe format.
	S370FPIBw.d Format (p. 148)	Writes positive integer binary (fixed-point) values in IBM mainframe format.
	S370FRBw.d Format (p. 149)	Writes real binary (floating-point) data in IBM mainframe format.
	S370FZDw.d Format (p. 151)	Writes zoned decimal data in IBM mainframe format.
	S370FZDLw.d Format (p. 152)	Writes zoned decimal leading–sign data in IBM mainframe format.
	S370FZDSw.d Format (p. 153)	Writes zoned decimal separate leading-sign data in IBM mainframe format.
	S370FZDTw.d Format (p. 154)	Writes zoned decimal separate trailing-sign data in IBM mainframe format.
	S370FZDUw.d Format (p. 155)	Writes unsigned zoned decimal data in IBM mainframe format.
	SSNw. Format (p. 156)	Writes Social Security numbers.
	VAXRBw.d Format (p. 162)	Writes real binary (floating-point) data in VMS format.
	VMSZNw.d Format (p. 163)	Generates VMS and MicroFocus COBOL zoned numeric data.

Category	Language Elements	Description
	w.d Format (p. 164)	Writes standard numeric data one digit per byte.
	WORDFw. Format (p. 177)	Writes numeric values as words with fractions that are shown numerically.
	WORDSw. Format (p. 178)	Writes numeric values as words.
	Zw.d Format (p. 193)	Writes standard numeric data with leading 0s.
	ZDw.d Format (p. 194)	Writes numeric data in zoned decimal format .

Dictionary

$ASCII*w*. Format

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.

Example

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

Value of x	Result*
abc	616263
ABC	414243
();	28293B

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

$BASE64X*w*. Format

Converts character data into ASCII text by using Base 64 encoding.

Category: Character

Alignment: left

Syntax

$BASE64X*w*.

Syntax Description

w

specifies the width of the output field.

Default: 1

Range: 1-32767

Details

Base 64 is an industry encoding method whose encoded characters are determined by using a positional scheme that uses only ASCII characters. Several Base 64 encoding schemes have been defined by the industry for specific uses, such as e-mail or content masking. SAS maps positions 0 - 61 to the characters A - Z, a - z, and 0 - 9. Position 62 maps to the character +, and position 63 maps to the character /.

The following are some uses of Base 64 encoding:

- embed binary data in an XML file
- encode passwords
- encode URLs

The '=' character in the encoded results indicates that the results have been padded with zero bits. In order for the encoded characters to be decoded, the '=' must be included in the value to be decoded.

Example

put x $base64x64.;

Value of x	Result
"FCA01A7993BC"	RkNBMDFBNzk5M0JD
"MyPassword"	TXlQYXNzd29yZA==
"www.mydomain.com/ myhiddenURL"	d3d3Lm15ZG9tYWluLmNvbi9teWhpZGRlblVSTA==

See Also

- The LIBNAME statement option "XMLDOUBLE=DISPLAY | INTERNAL" in Chapter 8 of *SAS XML LIBNAME Engine: User's Guide*

Informats:

- "$BASE64Xw. Informat" on page 225

$BINARY*w*. Format

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.

Example

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

Value of `name`	Result
ASCII	EBCDIC

Value of `name`	Result	
AB	0100000101000010	1100000111000010

$CHAR*w*. Format

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. Alternatively, 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)

Example

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

Value of `name`	Result
	----+----1
XYZ	XYZ

$EBCDIC*w*. Format

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.

Example

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

Value of `name`	Result *
qrs	9899A2
QRS	D8D9E2
+;>	4E5E6E

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

$HEX*w*. Format

Converts character data to hexadecimal representation.

Category: Character

Alignment: left

See: “$HEXw. Format: UNIX” in *SAS Companion for UNIX Environments*
“$HEXw. Format: Windows” in *SAS Companion for Windows*

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

Tips:

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.

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 characters. Each blank counts as one character, including trailing blanks.

Comparisons

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

Example

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

Value of `name`	Result	
	EBCDIC	ASCII
	`----+----1`	`----+----1`
`AB`	`C1C2`	`4142`

$MSGCASE*w*. Format

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: 1, if the length of the variable is undefined. Otherwise, the default is 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.

Example

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

Value of `name`	Result
`sas`	`SAS`

See Also

System Options:

- "MSGCASE System Option: UNIX" in *SAS Companion for UNIX Environments*
- "MSGCASE System Option: Windows" in *SAS Companion for Windows*
- "MSGCASE System Option: z/OS" in *SAS Companion for z/OS*

$N8601B*w.d* Format

Writes ISO 8601 duration, datetime, and interval forms by using the basic notations P*n*Y*n*M*n*DT*n*H*n*M*n*S and *yyyymmdd*T*hhmmss*.

Categories: Date and Time

ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.4, complete representation

Syntax

$N8601B*w*.*d*

Syntax Description

w

specifies the width of the output field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601B format writes ISO 8601 duration, datetime, and interval values as character data for the following basic notations:

- P*n*Y*n*M*n*DT*n*H*n*M*n*S
- *yyyymmdd*T*hhmmss*
- P*n*Y*n*M*n*DT*n*H*n*M*n*S/*yyyymmdd*T*hhmmss*
- *yyyymmdd*T*hhmmss*T/P*n*Y*n*M*n*DT*n*H*n*M*n*S

The lowest-order component can contain fractions, as in these examples:

- **`p2y3.5m`**
- **`p00020304T05.335`**

Example

```
put nb $n8601b.;
```

Value of nb	Result
`0002405050112FFC`	`P2Y4M5DT5H1M12S`
`2012915155300FFD`	`20120915T155300`
`2012915000000FFD2014915000000FFD`	`20120915T000000/20140915T000000`

Value of nb	Result
`0033104030255FFC2012915155300FFD`	`P33Y1M4DT3H2M55S/20120915T155300`

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

$N8601BA*w.d* Format

Writes ISO 8601 duration, datetime, and interval forms by using the basic notations P*yyyymmdd*T*hhmmss* and *yyyymmdd*T*hhmmss*.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: IS0 8601 Element 5.5.4.2, alternative format

Syntax

$N8601BA*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601BA format writes ISO 8601 duration, datetime, and interval values as character data for the following basic notations:

- P*yyyymmdd*T*hhmmss*
- *yyyymmdd*T*hhmmss*
- P*yyyymmdd*T*hhmmss*/*yyyymmdd*T*hhmmss*
- *yyyymmdd*T*hhmmss*/P*yyyymmdd*T*hhmmss*

The lowest-order component can contain fractions, as in these examples:

- **p00023.5**
- **00020304T05.335**

Example

```
put @1 nba $N8601ba.;
```

Value of nba	Result
00024050501127D0	P00020405T050112.5
2012915155300FFD	20120915T155300
00023040506075282012915155300FFD	P00020304T050607.33/20120915T155300

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

$N8601E*w.d* Format

Writes ISO 8601 duration, datetime, and interval forms by using the extended notations P*n*Y*n*M*n*DT*n*H*n*M*n*S and *yyyy-mm-dd*T*hh:mm:ss*.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.4, complete representation

Syntax

$N8601E*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601E format writes ISO 8601 duration, datetime, and interval values as character data for the following basic notations:

- P*n*Y*n*M*n*DT*n*H*n*M*n*S
- *yyyy-mm-dd*T*hh:mm:ss*
- P*n*Y*n*M*n*DT*n*H*n*M*n*S/*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*T/P*n*Y*n*M*n*DT*n*H*n*M*n*S

The lowest-order component can contain fractions, as in these examples:

- **`p2y3.5m`**
- **`p0002-03-04T05.335`**

Example

```
put @1 ne $n8601e.;
```

Value of ne	Result
00024050501127D0	P2Y4M5DT5H1M12.5S
2012915155300FFD	2012-09-15T15:53:00
2012915000000FFD2014915000000FFD	2012-09-15T00:00:00/2013-09-15T00:00:00
0033104030255FFC2012915155300FFD	P33Y1M4DT3H2M55S/2012-09-15T15:53:00

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

$N8601EA*w.d* Format

Writes ISO 8601 duration, datetime, and interval forms by using the extended notations P*yyyy-mm-dd*T*hh:mm:ss* and *yyyy-mm-dd*T*hh:mm:ss*.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.4, complete representation

Syntax

$N8601EA*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601EA format writes ISO 8601 duration, datetime, and interval values as character data for the following basic notations:

- P*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*
- P*yyyy-mm-dd*T*hh:mm:ss*/*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*/P*yyyy-mm-dd*T*hh:mm:ss*

The lowest-order component can contain fractions, as in these examples:

- **p00023.5**
- **0002-03-04T05.335**

Example

```
put @1 nea $N8601ea.;
```

Value of nea	Result
00024050501127D0	P0002-04-05T05:01:12.500
2012915155300FFD	2012-09-15T15:53:00
00023040506075282012915155300FFD	P0002-03-04T05:06:07.330/2012-09-15T15:53:00

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

$N8601EH*w.d* Format

Writes ISO 8601 duration, datetime, and interval forms by using the extended notations P*yyyy-mm-dd*T*hh:mm:ss* and *yyyy-mm-dd*T*hh:mm:ss*, using a hyphen (-) for omitted components.

Categories: Date and Time
ISO 8601

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.4, complete representation

Syntax

$N8601EH*w*.*d*

Syntax Description

w

specifies the width of the output field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601EH format writes ISO 8601 duration, datetime, and interval values as character data, using a hyphen (-) to represent omitted components, for the following extended notations:

- P*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*
- P*yyyy-mm-dd*T*hh:mm:ss*/*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*/P*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*/*yyyy-mm-dd*T*hh:mm:ss*

Omitted datetime components are always displayed; they are never truncated.

Example

```
put a $n8601eh.;
```

Value of a	Result
00023FFFFFFFFFFC2012FFF15FFFFFFD	P0002-03—T-:-:-/2012——T15:-:-
2012FFF15FFFFFFdFFFF3FF1553FFFFC	2012--T15:-:-/P—03—T15:53:-

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

$N8601EX*w.d* Format

Writes ISO 8601 duration, datetime, and interval forms by using the extended notations P*yyyy-mm-dd*T*hh:mm:ss* and *yyyy-mm-dd*T*hh:mm:ss*, using an x for each digit of an omitted component.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Elements 5.5.3, 5.5.4.1, and 5.5.4.2

Syntax

$N8601EX*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601EX format writes ISO 8601 duration, datetime, and interval values as character data, using a hyphen (-) to represent omitted components, for the following extended notations:

- P*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*
- P*yyyy-mm-dd*T*hh:mm:ss*/*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*/P*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*/*yyyy-mm-dd*T*hh:mm:ss*

Omitted datetime components are always displayed; they are never truncated.

Example

```
put nex $n8601ex.;
```

Value of nex	Result
00023FFFFFFFFFFC2012FFF15FFFFFFD	P0002-03xxTxx:xx:xx/2012-xx-xxT15:xx:xx
2012FFF15FFFFFFdFFFF3FF1553FFFFC	2012-xx-xxT15:xx:xx/Pxxxx-03-xxT15:53:xx

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

$N8601H*w.d* Format

Writes ISO 8601 duration, datetime, and interval forms P*n*Y*n*M*n*DT*n*H*n*M*n*S and *yyyy-mm-dd*T*hh:mm:ss*, dropping omitted components in duration values and using a hyphen (-) for omitted components in datetime values.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Elements 5.5.3, 5.5.4.1, and 5.5.4.2

Syntax

$N8601H*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601H format writes ISO 8601 durations, intervals, and datetimes in the following forms, omitting components in the P*n*Y*n*M*n*DT*n*H*n*M*n*S form and using a hyphen (-) to represent omitted components in the datetime form:

- P*n*Y*n*M*n*DT*n*H*n*M*n*S
- *yyyy-mm-dd*T*hh:mm:ss*
- P*n*Y*n*M*n*DT*n*H*n*M*n*S/*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*T/P*n*Y*n*M*n*DT*n*H*n*M*n*S
- *yyyy-mm-dd*T*hh:mm:ss*/*yyyy-mm-dd*T*hh:mm:ss*

Omitted datetime components are always displayed; they are never truncated.

Example

```
put nh $n8601h.;
```

Value of nh	Result
`0002304FFFFFFFFC2012FFF15FFFFFFD`	`P2Y3M4D/2012—T15:-:-`
`FFFF102FFFFFFFFD2012FFF15FFFFFFD`	`—01-02T-:-:-0/2012—T15:-:-`

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

$N8601X*w.d* Format

Writes ISO 8601 duration, datetime, and interval forms P*n*Y*n*M*n*DT*n*H*n*M*n*S and *yyyy-mm-dd*T*hh:mm:ss*, dropping omitted components in duration values and using an x for each digit of an omitted component in datetime values.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Elements 5.5.3, 5.5.4.1, and 5.5.4.2

Syntax

$N8601X*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601X format writes ISO 8601 durations, intervals, and datetimes in the following forms, omitting components in the P*n*Y*n*M*n*DT*n*H*n*M*n*S form and using an x to represent omitted components in the datetime form:

- P*n*Y*n*M*n*DT*n*H*n*M*n*S
- *yyyy-mm-dd*T*hh:mm:ss*
- P*n*Y*n*M*n*DT*n*H*n*M*n*S/*yyyy-mm-dd*T*hh:mm:ss*
- *yyyy-mm-dd*T*hh:mm:ss*T/P*n*Y*n*M*n*DT*n*H*n*M*n*S
- *yyyy-mm-dd*T*hh:mm:ss*/*yyyy-mm-dd*T*hh:mm:ss*

Omitted datetime components are always displayed; they are never truncated.

Example

```
put nx $n8601x.;
```

Value of nx	Result
`0002304FFFFFFFFC2011FFF15FFFFFFD`	`P2Y3M4D/2011-xx-xxT15:xx:xx`
`FFFF102FFFFFFFFD2011FFF15FFFFFFd`	`xxxx-01-02Txx:xx:xx/2011x-xxT15:xx:xx`

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

$OCTAL*w*. Format

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* by 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.

Example

The following example shows ASCII output when you use the $OCTAL*w*. format.

```
data _null_;
   infile datalines truncover;
   input item $5.;
   put item $octal15.;
   datalines;
art
rice
bank
;
run;
```

SAS writes the following results to the log.

```
141162164040040
162151143145040
142141156153040
```

$QUOTE*w*. Format

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: 2, if the length of the variable is undefined. Otherwise, the default is the length of the variable + 2

Range: 2–32767

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

Details

The following list describes the output that SAS produces when you use the $QUOTE*w*. format. For examples of these items, see the examples below.

- If your data value is not enclosed in quotation marks, SAS encloses the output in double quotation marks.
- If your data value is not enclosed in quotation marks, but the value contains a single quotation mark, SAS does the following:
  - encloses the data value in double quotation marks
  - does not change the single quotation mark
- If your data value begins and ends with single quotation marks, and the value contains double quotation marks, SAS does the following:
  - encloses the data value in double quotation marks
  - duplicates the double quotation marks that are found in the data value
  - does not change the single quotation marks
- If your data value begins and ends with single quotation marks, and the value contains two single contiguous quotation marks, SAS does the following:
  - encloses the value in double quotation marks
  - does not change the single quotation marks
- If your data value begins and ends with single quotation marks, and contains both double quotation marks and single, contiguous quotation marks, SAS does the following:
  - encloses the value in double quotation marks
  - duplicates the double quotation marks that are found in the data value
  - does not change the single quotation marks
- If the length of the target field is not large enough to contain the string and its quotation marks, SAS returns as much of the quoted string that will fit in the field.

Example

```
put name $quote20.;
```

Value of `name`	Result
	`----+----1----+----2`
`SAS`	`"SAS"`
`SAS's`	`"SAS's"`
`'ad"verb"'`	`"'ad""verb""'"`
`'ad''verb'`	`"'ad''verb'"`
`'"ad"''"verb"'`	`----+----1----+----2` `"'""ad""''""verb""'"`

Value of `name`	Result
`deoxyribonucleotide`	`"deoxyribonucleotid"` *

* deoxyribonucleotide is 19 characters. When SAS adds the quotation marks, the length of the string is 21 characters. SAS truncates the letter e at the end of the text to accommodate the quotation marks.

$REVERJ*w*. Format

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 *w* is not specified

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.

Example

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

Name*	Result
	`----+----1`
`ABCD###`	`   DCBA`
`###ABCD`	`DCBA`

* The character # represents a blank space.

$REVERS*w*. Format

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 if *w* is not specified

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.

Example

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

Name*	Result
	----+----1
ABCD###	DCBA
###ABCD	DCBA

* The character # represents a blank space.

$UPCASE*w*. Format

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 default is the length of the variable

Range: 1–32767

Details

Special characters, such as hyphens and other symbols, are not altered.

Example

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

Value of `name`	Result
	`----+----1`
`coxe-ryan`	`COXE-RYAN`

$VARYING*w*. Format

Writes character data of varying length.

Valid in: 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 default is 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.

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

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

Tips:

If the value of *length-variable* is 0, negative, 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*.

If *length-variable* is greater than or equal to *w*, SAS writes *w* columns.

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.

Value of name *	Result
	`----+----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 *	Result
	`----+----1----+----2----+`
`New York`	`New York`
`Toronto`	`Toronto`
`Buenos Aires`	`Buenos Aires`

Values *	Result
`Tokyo`	`Tokyo`

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

$*w*. Format

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 default is 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, or use list output with the colon (:) format modifier and the format of your choice.

Example

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

Value of `name` *	Result
	`----+----1----+----2`
`#Cary`	` Cary`
`Tokyo`	`Tokyo`

* The character # represents a blank space.

BEST*w*. Format

SAS chooses the best notation.

Category: Numeric

Alignment: right

See: "BESTw. Format: z/OS" in *SAS Companion for z/OS*

Syntax

BEST*w*.

Syntax Description

w

specifies the width of the output field.

Default: 12

Range: 1–32

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

Details

When a format is not specified for writing a numeric value, SAS uses the BEST*w*. format as the default format. The BEST*w*. format writes numbers as follows:

- Values are written with the maximum precision, as determined by the width.
- Integers are written without decimals.
- Numbers with decimals are written with as many digits to the left and right of the decimal point as needed or as allowed by the width.
- Values that can be written within the given width are written without trailing zeros.
- Values that cannot be written within the given width are written with the maximum allowable number of decimal places as determined by the width.
- Extreme values might be written in scientific notation.

SAS stores the complete value regardless of the format that is used.

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. Integers print without a decimal.
- The D*w.p* format writes numbers with the desired precision and more alignment than the BEST*w* format.
- The BESTD*w.p* format is a combination of the BEST*w*. format and the D*w.p* format in that it formats all numeric data, and it does a better job of aligning decimals than the BEST*w*. format.

- The *w.d* format aligns decimal points, if possible, but does not necessarily show the same precision for all numbers.

Example

The following statements produce these results.

SAS Statements	Result
	`----+----1`
`x=1257000;` `put x best6.;`	`1.26E6`
`x=1257000;` `put x best3.;`	`1E6`

See Also

Formats:

- "BESTDw.p Format" on page 58

BESTD*w.p* Format

Prints numeric values, lining up decimal places for values of similar magnitude, and prints integers without decimals.

Category: Numeric

Alignment: right

Syntax

BESTD*w.p*

Syntax Description

w

specifies the width of the output field.

Default: 12

Range: 1–32

p

specifies the precision. This argument is optional.

Default: 3

Range: 0 to *w*–1

Requirement: must be less than *w*

Tip: If *p* is omitted or is specified as 0, then *p* is set to 3.

Details

The BESTD*w.p* format writes numbers so that the decimal point aligns in groups of values with similar magnitude. Integers are printed without a decimal point. Larger values of *p* print the data values with more precision and potentially more shifts in the decimal point alignment. Smaller values of *p* print the data values with less precision and a greater chance of decimal point alignment.

The format chooses the number of decimal places to print for ranges of values, even when the underlying values can be represented with fewer decimal places.

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. Integers print without a decimal.
- The D*w.p* format writes numbers with the desired precision and more alignment than the BEST*w* format.
- The BESTD*w.p* format is a combination of the BEST*w*. format and the D*w.p* format in that it formats all numeric data, and it does a better job of aligning decimals than the BEST*w*. format.
- The *w.d* format aligns decimal points, if possible, but it does not necessarily show the same precision for all numbers.

Example

```
put x bestd14.;
```

Data Line	Result
	`----+----1----+`
`12345`	`12345`
`123.45`	`123.4500000`
`1.2345`	`1.2345000`
`.12345`	`0.1234500`
`1.23456789`	`1.2345679`

See Also

Formats:

- “BESTw. Format” on page 57
- “Dw.p Format” on page 71

BINARY*w*. Format

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

Comparisons

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

Example

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

Value of x	Result
	----+----1
123.45	01111011
123	01111011
-123	10000101

B8601DA*w*. Format

Writes date values by using the ISO 8601 basic notation *yyyymmdd*.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

B8601DA*w*.

Syntax Description

w
specifies the width of the output field.

Default: 10

Range: 8–10

Details

The B8601DA format writes the date value by using the ISO 8601 basic date notation *yyyymmdd*:

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 0 and 31.

Example

```
put bda b8601da.;
```

Value of bda	Result
18885	20110915
18628	20110101

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

B8601DN*w*. Format

Writes dates from datetime values by using the ISO 8601 basic notation *yyyymmdd*.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

B8601DN*w*.

Syntax Description

w

specifies the width of the output field.

Default: 10

Range: 8–10

Details

The B8601DN format writes the date from a datetime value by using the ISO 8601 basic date notation *yyyymmdd*:

yyyy
: is a four-digit year.

mm
: is a two-digit month (zero padded) between 01 and 12.

dd
: is a two-digit day of the month (zero padded) between 01 and 31.

Example

```
put bdn b8601dn.;
```

Value of bdn	Result
1631664000	20110915

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

B8601DT*w.d* Format

Writes datetime values by using the ISO 8601 basic notation *yyyymmdd*T*hhmmss<ffffff>*.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

B8601DT*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 19

Range: 15–26

d

specifies the number of digits to the right of the seconds value that represents a fraction of a second. This argument is optional.

Default: 0

Range: 0–6

Details

The B8601DT format writes the datetime value by using the ISO 8601 basic datetime notation *yyyymmdd*T*hhmmss*<*ffffff*>:

yyyy
: is a four-digit year.

mm
: is a two-digit month (zero padded) between 01 and 12.

dd
: is a two-digit day of the month (zero padded) between 01 and 31.

hh
: is a two-digit hour (zero padded) between 00 and 23.

mm
: is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

ffffff
: are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```
put bdt b8601dt.;
```

Value of bdt	Result
---- +----1	
1631721180	20110915T155300

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

B8601DZ*w.* Format

Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone basic notation *yyyymmdd*T*hhmmss*+0000.

Categories: Date and Time
ISO 8601

Alignment: left

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

B8601DZ*w*.

Syntax Description

w

specifies the width of the output field.

Default: 26

Range: 20–35

Details

UTC values specify a time and a time zone based on the zero meridian in Greenwich, England. The B8601DZ format writes SAS datetime values for the zero meridian date and time by using one of the following ISO 8601 basic datetime notations:

- *yyyymmdd*T*hhmmss*+0000

  Note: Use this form when *w* is large enough to support this time zone notation.

- *yyyymmdd*T*hhmmss*Z

  Note: Use this form when *w* is not large enough to support the +0000 time zone notation.

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

hh
is a two-digit hour (zero padded) between 00 and 23.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
is a two-digit second (zero padded) between 00 and 59.

+0000
indicates the UTC time for the zero meridian (Greenwich, England).

An ISO 8601 time or datetime value that specifies a time zone offset is adjusted by the number of hours and minutes that is specified in the offset and processed as the time or datetime for the zero meridian (Greenwich, England). The B8601DZ format always writes the datetime value using the zero meridian offset value of +0000. To write a datetime that uses the UTC offset other than +0000, see "B8601LZw. Format" on page 65.

Restriction: The shorter form +00 is not supported.

Z

indicates that the time is for the zero meridian (Greenwich, England) or +0000 UTC time. Z is used when the width of the format does not support the +0000 notation.

Example

```
put bdz b8601dz20.;
```

Datetime Value	Value of bdz	Result
20110915T155300+0500	1631703180*	20110915T105300+0000
20110915T155300Z	1631721180	20110915T155300+0000

* The ISO 8601 value specifies a time zone offset of five hours. When SAS read the value, the SAS datetime value was adjusted by five hours. The Result column shows the adjustment of five hours.

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

B8601LZ*w.* Format

Writes time values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic time notation *hhmmss*+|–*hhmm*.

Categories: Date and Time
ISO 8601

Alignment: left

Supports: ISO 8601 Elements 5.3.3 and 5.3.4.2

Syntax

B8601LZ*w*.

Syntax Description

w

specifies the width of the output field.

Default: 14

Range: 9–20

Details

The B8601LZ format writes time values without making any adjustments, and appends the UTC time zone offset for the local SAS session by using the ISO 8601 basic notation *hhmmss*+|–*hhmm*:

hh
: is a two-digit hour (zero padded) between 00 and 23.

mm
: is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

+|–*hhmm*
: is an hour and minute signed offset from zero meridian time. Note that the offset must be +|–*hhmm* (that is, + or – and four characters).

  Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

  Restriction: The shorter form +|–*hh* is not supported.

When SAS reads a UTC time by using the B8601TZ informat, and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 000000 and 235959. If the B8601LZ format attempts to format a time outside of this time range, the time is formatted with asterisks to indicate that the value is out of range.

Example

This PUT statement writes the time for the Eastern Standard time zone:

```
put blz b8601lz.;
```

Value of blz	Result
46380	125300-0500

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

B8601TM*w.d* Format

Writes time values by using the ISO 8601 basic notation *hhmmss<ffff>*.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.3.1.1, complete representation

Syntax

B8601TM*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 6–15

d

specifies the number of digits to the right of the seconds value that represents a fraction of a second. This argument is optional.

Default: 0

Range: 0–6

Details

The B8601TM format writes SAS time values by using the ISO 8601 basic time notation *hhmmss<ffffff>*:

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

ffffff

are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```
put btm b8601tm.;
```

Value of btm	Result
57180	155300

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

B8601TZ*w.* Format

Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 basic time notation *hhmmss*+|–*hhmm*.

Categories: Date and Time
ISO 8601

Alignment: left

Supports: ISO 8601 Elements 5.3.3 and 5.3.4

Syntax

B8601TZ*w*.

Syntax Description

w
specifies the width of the output field.

Default: 14

Range: 9–20

Details

UTC time values specify a time and a time zone based on the zero meridian in Greenwich, England. The B8601TZ format adjusts the time value to be the time at the zero meridian and writes the time value in one of the following ISO 8601 basic time notations:

- *hhmmss*+|–*hhmm*

  Note: Use this form when *w* is large enough to support this time notation.

- *hhmmss*Z

  Note: Use this form when *w* is not large enough to support the +|–*hhmm* time zone notation.

hh
is a two-digit hour (zero padded) between 00 and 23.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
is a two-digit second (zero padded) between 00 and 59.

+|–*hh:mm*
is an hour and minute signed offset from zero meridian time. Note that the offset must be +|–*hhmm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Z
indicates that the time is for zero meridian (Greenwich, England) or +0000 UTC time.

When SAS reads a UTC time by using the B8601TZ informat, and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 000000 and 240000. If the B8601TZ format attempts to format a time outside of this time range, the time is formatted with asterisks to indicate that the value is out of range.

Comparisons

For time values between 000000 and 240000, the B8601TZ format adjusts the time value to be the time at the zero meridian and writes the time value in the international standard extended time notation. The B8601LZ format makes no adjustment to the time and writes time values in the international standard extended time notation, using a UTC time zone offset for the local SAS session.

Example

```
put btz b8601tz.;
```

Values for btz	Result
73441	202401+0000

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

COMMA*w.d* Format

Writes numeric values with a comma that separates every three digits and a period that separates the decimal fraction.

Category: Numeric

Alignment: right

Syntax

COMMA*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 6

Range: 1–32

Tip: Make *w* wide enough to write the numeric values, the commas, and the optional decimal point.

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Range: 0–31

Requirement: must be less than *w*

Details

The COMMA*w.d* format writes numeric values with a comma that separates 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.

Example

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

Value of `sales`	Result
	`----+----1----+----2`
`23451.23`	`        23,451.23`
`123451.234`	`       123,451.23`

See Also

Formats:

- "COMMAXw.d Format" on page 70
- "DOLLARw.d Format" on page 81

COMMAX*w.d* Format

Writes numeric values with a period that separates every three digits and a comma that separates the decimal fraction.

Category: Numeric

Alignment: right

Syntax

COMMAX*w.d*

Syntax Description

w

specifies the width of the output field. This argument is optional.

Default: 6

Range: 1–32

Tip: Make *w* wide enough to write the numeric values, the commas, and the optional decimal point.

d

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 a period that separates 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.

Example

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

Value of `sales`	Result
	`----+----1----+----2`
`23451.23`	`    23.451,23`
`123451.234`	`   123.451,23`

D*w.p* Format

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

Category: Numeric

Alignment: right

Syntax

D*w.p*

Syntax Description

w

specifies the width of the output field. This argument is optional.

Default: 12

Range: 1–32

p

specifies the precision. This argument is optional.

Default: 3

Range: 0–9

Requirement: *p* must be less than *w*

Tips:

If *p* is omitted or is specified as 0, then *p* is set to 3.

If zero is the desired precision, use the *w.d* format in place of the D*w.p* format.

Details

The D*w.p* format writes numbers so that the decimal point aligns in groups of values with similar magnitude. Larger values of *p* print the data values with more precision and potentially more shifts in the decimal point alignment. Smaller values of *p* print the data values with less precision and a greater chance of decimal point alignment.

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.p* writes numbers with the desired precision and more alignment than the BEST*w* format.
- The BESTD*w.p* format is a combination of the BEST*w*. format and the D*w.p* format in that it formats all numeric data, and it does a better job of aligning decimals than the BEST*w*. format.
- The *w.d* format aligns decimal points, if possible, but it does not necessarily show the same precision for all numbers.

Example

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

Value of x	Result
	`----+----1`
12345	`  12345.0`
1234.5	`   1234.5`
123.45	`123.45000`
12.345	` 12.34500`
1.2345	`  1.23450`
.12345	`  0.12345`

See Also

Formats:

• “BESTDw.p Format” on page 58

DATE*w*. Format

Writes date values in the form *ddmmmyy*, *ddmmmyyyy*, or *dd-mmm-yyyy*.

Category: Date and Time

Alignment: right

Syntax

DATE*w*.

Syntax Description

w

specifies the width of the output field.

Default: 7

Range: 5–11

Tip: Use a width of 9 to print a 4-digit year without a separator between the day, month, and year. Use a width of 11 to print a 4-digit year using a hyphen as a separator between the day, month, and year

Details

The DATE*w*. format writes SAS date values in the form *ddmmmyy*, *ddmmmyyyy*, or *dd-mmm-yyyy*, 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-digit or four-digit integer that represents the year.

Example

The example table uses the input value of 19068, which is the SAS date value that corresponds to March 16, 2012.

SAS Statement	Result
	`----+----1----+`
`put day date5.;`	`16MAR`
`put day date6.;`	` 16MAR`
`put day date7.;`	`16MAR12`
`put day date8.;`	` 16MAR12`

SAS Statement	Result
`put day date9.;`	`16MAR2012`
`put day date11.;`	`16-MAR-2012`

See Also

Functions:

- "DATE Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267

DATEAMPM*w.d* Format

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

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Range: 0–39

Requirement: must be less than *w*

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 DATETIME*w.d* format except that DATEAMPM*w.d* prints AM or PM at the end of the time.

Example

The example table uses the input value of 1650538894, which is the SAS datetime value that corresponds to 11:01:34 a.m. on April 20, 2012.

SAS Statement	Result
	`----+----1----+----2----+`
`put event dateampm.;`	`20APR12:11:01:34 AM`
`put event dateampm7.;`	`20APR12`
`put event dateampm10.;`	`20APR:11`
`put event dateampm13.;`	`20APR12:11 AM`
`put event dateampm22.2;`	`20APR12:11:01:34.00 AM`

See Also

Formats:

- “DATETIMEw.d Format” on page 75

DATETIME*w.d* Format

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

Category: Date and Time

Alignment: right

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

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* and a value to *d* to return values with optional decimal fractions of seconds.

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Range: 0–39

Requirement: must be less than *w*

Details

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

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 in 24–hour clock time.

mm
: is an integer that represents the minutes.

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

Example

The example table uses the input value of 1668138559, which is the SAS datetime value that corresponds to 3:49:19 a.m. on November 10, 2012.

SAS Statement	Result
	`----+----1----+----2----+`
`put event datetime.;`	`10NOV12:03:49:19`
`put event datetime7.;`	`10NOV12`
`put event datetime12.;`	`   10NOV12:03`

SAS Statement	Result
`put event datetime18.;`	`  10NOV12:03:49:19`
`put event datetime18.1;`	`10NOV12:03:49:19.0`
`put event datetime19.;`	` 10NOV2012:03:49:19`
`put event datetime20.1;`	`10NOV2012:03:49:19.0`
`put event datetime21.2;`	`10NOV2012:03:49:19.00`

See Also

Formats:

- “DATEw. Format” on page 73
- “TIMEw.d Format” on page 157

Functions:

- “DATETIME Function” in *SAS Functions and CALL Routines: Reference*

Informats:

- “DATEw. Informat” on page 267
- “DATETIMEw. Informat” on page 268
- “TIMEw. Informat” on page 332

DAY*w*. Format

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

Example

The example table uses the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

SAS Statement	Result
	`----+`
`put date day2.;`	`14`

DDMMYY*w*. Format

Writes date values in the form *ddmm<yy>yy* or *dd/mm/<yy>yy*, where a forward slash is the separator and the year appears as either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

DDMMYY*w*.

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 2–10

Interaction: When *w* has a value of from 2 to 5, the date appears with as much of the day and the month as possible. When *w* is 7, the date appears as a two-digit year without slashes.

Details

The DDMMYY*w*. format writes SAS date values in the form *ddmm<yy>yy* or *dd/mm/<yy>yy*:

dd

is an integer that represents the day of the month.

/

is the separator.

mm

is an integer that represents the month.

<yy>yy

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

Example

The following examples use the input value of 19351, which is the SAS date value that corresponds to December 24, 2012.

SAS Statement	Result
	`----+----1`
`put date ddmmyy5.;`	`24/12`
`put date ddmmyy6.;`	`241212`
`put date ddmmyy7.;`	` 241212`
`put date ddmmyy8.;`	`24/12/12`
`put date ddmmyy10.;`	`24/12/2012`

See Also

Formats:

- "DATEw. Format" on page 73
- "DDMMYYxw. Format" on page 79
- "MMDDYYw. Format" on page 113
- "YYMMDDw. Format" on page 181

Functions:

- "MDY Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267
- "DDMMYYw. Informat" on page 270
- "MMDDYYw. Informat" on page 292
- "YYMMDDw. Informat" on page 347

DDMMYY*xw*. Format

Writes date values in the form *ddmm<yy>yy* or *dd-mm-yy<yy>*, where the *x* in the format name is a character that represents the special character that separates the day, month, and year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

DDMMYY*xw*.

Syntax Description

x

identifies a separator or specifies that no separator appear between the day, the month, and the year. The following are valid values for *x*:

B
: separates with a blank

C
: separates with a colon

D
: separates with a hyphen

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

Interactions:

When *w* has a value of from 2 to 5, the date appears with as much of the day and the month as possible. When *w* is 7, the date appears as a two-digit year without separators.

When *x* has a value of N, the width range changes to 2–8.

Details

The DDMMYY*xw*. format writes SAS date values in the form *ddmm<yy>yy* or *ddxmmx<yy>yy*:

dd

is an integer that represents the day of the month.

x

is a specified separator.

mm

is an integer that represents the month.

<yy>yy

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

Example

The following examples use the input value of 19137, which is the SAS date value that corresponds to May 24, 2012.

SAS Statement	Result
	`----+----1----+`

SAS Statement	Result
`put date ddmmyyc5.;`	`24:05`
`put date ddmmyyd8.;`	`24-05-12`
`put date ddmmyyp10.;`	`24.05.2012`
`put date ddmmyyn8.;`	`24052012`

See Also

Formats:

- "DATEw. Format" on page 73
- "DDMMYYw. Format" on page 78
- "MMDDYYxw. Format" on page 115
- "YYMMDDxw. Format" on page 183

Functions:

- "DAY Function" in *SAS Functions and CALL Routines: Reference*
- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DDMMYYw. Informat" on page 270

DOLLAR*w.d* Format

Writes numeric values with a leading dollar sign, a comma that separates every three digits, and a period that separates the decimal fraction.

Category: Numeric

Alignment: right

Syntax

DOLLAR*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 6

Range: 2–32

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Range: 0–31

Requirement: must be less than *w*

Details

The DOLLAR*w.d* format writes numeric values with a leading dollar sign, 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 might 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. For more details, see Chapter 23, “FORMAT Procedure” in *Base SAS Procedures Guide*.

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.

Example

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

Value of `netpay`	Result
	`----+----1----+`
`1254.71`	`  $1,254.71`

See Also

Formats:

- “COMMAw.d Format” on page 69
- “DOLLARXw.d Format” on page 82

DOLLARX*w.d* Format

Writes numeric values with a leading dollar sign, a period that separates every three digits, and a comma that separates the decimal fraction.

Category: Numeric

Alignment: right

Syntax

DOLLARX*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 6

Range: 2–32

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Default: 0

Range: 0–31

Requirement: must be less than *w*

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 might 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 For details, see Chapter 23, "FORMAT Procedure" in *Base SAS Procedures Guide*.

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.

Example

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

Value of `netpay`	Result
	`----+----1----+`
`1254.71`	`    $1.254,71`

See Also

Formats:

- "COMMAXw.d Format" on page 70

- "DOLLARw.d Format" on page 81

DOWNAMEw. Format

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.

Example

The example table uses the input value of 19137, which is the SAS date value that corresponds to May 24, 2012.

SAS Statement	Result
	`----+----1`
`put date downame.;`	` Thursday`

See Also

Formats:

- "WEEKDAYw. Format" on page 168

DTDATEw. Format

Expects a datetime value as input and writes date values in the form *ddmmmyy* or *ddmmmyyyy*.

Category: Date and Time

Alignment: right

Syntax

DTDATE*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 DTDATE*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
: are the first three letters of the month name.

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

Comparisons

The DTDATE*w*. format produces the same type of output that the DATE*w*. format produces. The difference is that the DTDATE*w*. format requires a datetime value.

Example

The example table uses a datetime value of 16APR2012:10:00:00 as input, and prints both a two-digit and a four-digit year for the DTDATE*w*. format.

SAS Statement	Result
	`----+----1`
`put trip_date=dtdate.;`	`16APR12`
`put trip_date=dtdate9.;`	`16APR2012`

See Also

Formats:

- "DATEw. Format" on page 73

DTMONYY*w*. Format

Writes the date part of a datetime value as the month and year in the form *mmmyy* or *mmmyyyy*.

Category: Date and Time

Alignment: right

Syntax

DTMONYY*w*.

Syntax Description

w

specifies the width of the output field.

Default: 5

Range: 5–7

Details

The DTMONYY*w*. format writes SAS datetime values in the form *mmmyy* or *mmmyyyy*, where

mmm

is the first three letters of the month name.

yy or *yyyy*

is a two–digit or four-digit integer that represents the year.

Comparisons

The DTMONYY*w*. format and the MONYY*w*. format are similar in that they both write date values. The difference is that DTMONYY*w*. expects a datetime value as input, and MONYY*w*. expects a SAS date value.

Example

The example table uses as input the value 1665986932, which is the SAS datetime value that corresponds to October 16, 2012, at 06:08:52 a.m.

SAS Statement	Result
	`----+----1`
`put date dtmonyy.;`	`OCT12`
`put date dtmonyy5.;`	`OCT12`
`put date dtmonyy6.;`	` OCT12`
`put date dtmonyy7.;`	`OCT2012`

See Also

Formats:

- "DATETIMEw.d Format" on page 75
- "MONYYw. Format" on page 123

DTWKDATX*w*. Format

Writes the date part of a datetime value as the day of the week and the date in the form *day-of-week*, *dd month-name yy* (or *yyyy*).

Category: Date and Time

Alignment: right

Syntax

DTWKDATX*w*.

Syntax Description

w

specifies the width of the output field.

Default: 29

Range: 3–37

Details

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

day-of-week
: is either the first three letters of the day name or the entire day name.

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

month-name
: is either the first three letters of the month name or the entire month name.

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

Comparisons

The DTWKDATX*w*. format is similar to the WEEKDATX*w*. format in that they both write date values. The difference is that DTWKDATX*w*. expects a datetime value as input, and WEEKDATX*w*. expects a SAS date value.

Example

The example table uses as input the value 1665986932, which is the SAS datetime value that corresponds to October 16, 20012, at 06:08:52 a.m.

SAS Statement	Result
	`----+----1----+----2----+`
`put date dtwkdatx.;`	`      Tuesday, 16 October 2012`
`put date dtwkdatx3.;`	`Tue`
`put date dtwkdatx8.;`	`      Tue`
`put date dtwkdatx25.;`	`      Tuesday, 16 Oct 2012`

See Also

Formats:

- "DATETIMEw.d Format" on page 75
- "WEEKDATXw. Format" on page 167

DTYEAR*w*. Format

Writes the date part of a datetime value as the year in the form *yy* or *yyyy*.

Category: Date and Time

Alignment: right

Syntax

DTYEAR*w*.

Syntax Description

w

specifies the width of the output field.

Default: 4

Range: 2–4

Details

The DTYEAR*w*. format is similar to the YEAR*w*. format in that they both write date values. The difference is that DTYEAR*w*. expects a datetime value as input, and YEAR*w*. expects a SAS date value.

Example

The example table uses as input the value 1665986932, which is the SAS datetime value that corresponds to October 16, 2012, at 06:08:52 a.m.

SAS Statement	Result
	`----+----1`
`put date dtyear.;`	`2012`
`put date dtyear2.;`	`12`
`put date dtyear3.;`	` 12`
`put date year4.;`	`2012`

See Also

Formats:

- "DATETIMEw.d Format" on page 75
- "YEARw. Format" on page 179

DTYYQC*w*. Format

Writes the date part of a datetime value as the year and the quarter and separates them with a colon (:).

Category: Date and Time

Alignment: right

Syntax

DTYYQC*w*.

Syntax Description

w

specifies the width of the output field.

Default: 4

Range: 4–6

Details

The DTYYQC*w*. format writes SAS datetime values in the form *yy* or *yyyy*, followed by a colon (:) and the numeric value for the quarter of the year.

Example

The example table uses as input the value 1665986932, which is the SAS datetime value that corresponds to October 16, 2012, at 06:08:52 p.m..

SAS Statement	Result
	`----+----1`
`put date dtyyqc.;`	`12:4`
`put date dtyyqc4.;`	`12:4`
`put date dtyyqc5.;`	` 12:4`
`put date dtyyqc6.;`	`2012:4`

See Also

Formats:

- "DATETIMEw.d Format" on page 75

E*w*. Format

Writes numeric values in scientific notation.

Category: Numeric

Alignment: right

See: "Ew. Format: z/OS" in *SAS Companion for z/OS*

Syntax

E*w*.

Syntax Description

w

specifies the width of the output field. The output field can display up to 14 significant digits.

Default: 12

Range: 7–32

Details

When formatting values in scientific notation, the E format reserves the first column of the result for a minus sign and formats up to 14 significant digits.

Example

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

Value of x	Result
	----+----1
1257	1.257E+03
-1257	-1.257E+03

E8601DA*w*. Format

Writes date values by using the ISO 8601 extended notation *yyyy-mm-dd*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601DA

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

E8601DA*w*.

Syntax Description

w

specifies the width of the output field.

Default: 10

Requirement: The width of the output field must be 10.

Details

The E8601DA format writes a date by using the ISO 8601 extended notation *yyyy-mm-dd*:

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

Example

```
put eda e8601da.;
```

Value for eda	Result
19251	2012-09-15

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

E8601DN*w*. Format

Writes dates from SAS datetime values by using the ISO 8601 extended notation *yyyy-mm-dd*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601DN

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

E8601DN*w*.

Syntax Description

w
specifies the width of the input field.

Default: 10

Requirement: The width of the input field must be 10.

Details

The E8601DN format writes the date by using the ISO 8601 extended date notation *yyyy-mm-dd*:

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

Example

```
put edn e8601dn.;
```

Value for edn	Result
`1663308532`	`2012-09-15`

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

E8601DT*w.d* Format

Writes datetime values by using the ISO 8601 extended notation *yyyy-mm-dd*T*hh:mm:ss.ffffff*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601DT

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

E8601DT*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 19

Range: 19–26

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–6

Details

The E8601DT format writes datetime values by using the ISO 8601 extended datetime notation *yyyy-mm-dd*T*hh:mm:ss.ffffff*:

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

hh
: is a two-digit hour (zero padded) between 00 and 23.

mm
: is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

ffffff
: are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```
put edt e8601dt25.3.;
```

Value of edt	Result
1663343580.2	2012-09-15T15:53:00.234

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

E8601DZ*w.* Format

Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone extended notation *yyyy-mm-dd*T*hh:mm:ss*+00:00.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601DZ

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

E8601DZ*w*.

Syntax Description

w
: specifies the width of the output field.

Default: 26

Range: 20–35

Details

UTC values specify a time and a time zone based on the zero meridian in Greenwich, England. The E8601DZ format writes SAS datetime values by using one of the following ISO 8601 extended datetime notations:

- *yyyy-mm-dd*T*hh:mm:ss*+00:00

  Note: Use this form when *w* is large enough to support this time zone notation.

- *yyyy-mm-dd*T*hh:mm:ss*Z

  Note: Use this form when *w* is not large enough to support the +00:00 time zone notation.

yyyy
: is a four-digit year.

mm
: is a two-digit month (zero padded) between 01 and 12.

dd
: is a two-digit day of the month (zero padded) between 01 and 31.

hh
: is a two-digit hour (zero padded) between 00 and 24.

mm
: is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

+00:00
: indicates that the time is for zero meridian (Greenwich, England) time.

  An ISO 8601 time or datetime value that specifies a time zone offset is adjusted by the number of hours and minutes that is specified in the offset and processed as the time or datetime for the zero meridian (Greenwich, England). The E8601DZ format always writes the datetime value by using the zero meridian offset value of +00:00. To write a datetime that uses the UTC offset other than +00:00, see "E8601LZw. Format" on page 96.

  Restriction: The shorter form +00 is not supported.

Z
: indicates that the time is for zero meridian (Greenwich, England) or +00:00 UTC time. Z is used when the width of the format does not support the +00:00 notation.

Example

```
put edz e8601dz.;
```

Value of edz	Result
1663332780	2012-09-15T12:53:00+00:00

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

E8601LZ*w.* Format

Writes time values as local time, appending the Coordinated Universal Time (UTC) offset for the local SAS session, using the ISO 8601 extended time notation *hh:mm:ss*+|–*hh:mm*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601LZ

Supports: ISO 8601 Element 5.3.1.1, complete representation

Syntax

E8601LZ*w*.

Syntax Description

w

specifies the width of the output field.

Default: 14

Range: 9–20

Details

The E8601LZ format writes time values without making any adjustments, and appends the UTC time zone offset for the local SAS session by using one of the following ISO 8601 extended time notations:

- *hh:mm:ss*+|–*hh:mm*

  Note: Use this form when *w* is large enough to support this time notation.

- *hh:mm:ss*Z

  Note: Use this form when *w* is not large enough to support the +|– *hh:mm* time zone notation.

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

+|–*hh:mm*

is an hour and minute signed offset from zero meridian time. Note that the offset must be +|–*hh:mm* (that is, + or – and five characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +02:00 indicates a two-hour time difference to the east of the zero meridian, and –06:00 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Z
: indicates zero meridian (Greenwich, England) or +00:00 UTC time.

SAS writes the time value by using the form *hh:mm.ffffff*, and appends the time zone indicator +|–*hh:mm* based on the time zone offset from the zero meridian for the local SAS session, or Z. The Z time zone indicator is used for format lengths that are less than 14.

If the same time is written using both zone indicators, they indicate two different times based on the UTC. For example, if the local SAS session uses Eastern Standard Time in the U.S., and the time value is 45824, SAS would write 12:43:44–04:00 or 12:43:44Z. The time 12:43:44–04:00 is the time 16:43:44+00:00 at the zero meridian. The Z indicates that the time is the time at the zero meridian, or 12:43:44+00:00.

When SAS reads a UTC time by using the E8601TZ informat, and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 00:00:00 and 24:00:00. If the E8601LZ format attempts to format a time outside of this time range, the time is formatted with asterisks to indicate that the value is out of range.

Example

This PUT statement writes the time for the Eastern Standard Time zone:

```
put elz e8601lz.;
```

Value of elz	Result
46380	12:53:00-5:00

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

E8601TM*w.d* Format

Writes time values by using the ISO 8601 extended notation *hh:mm:ss.ffffff*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601TM

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.3.1.1, complete representation, and 5.3.1.3, representation of decimal fractions

Syntax

E8601TM*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 8–15

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–6

Details

The E8601TM format writes SAS time values by using the ISO 8601 extended time notation *hh:mm:ss.ffffff*:

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

.ffffff

are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```
put etm e8601tm.;
```

Value of etm	Result
57180	15:53:00

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

E8601TZ*w.d* Format

Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation *hh*:*mm*:*ss*+|–*hh*:*mm*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601TZ

Supports: ISO 8601 Element 5.3.1.1, complete representation

Syntax

E8601TZ*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 14

Range: 9–20

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–6

Details

UTC time values specify a time and a time zone based on the zero meridian in Greenwich, England. The E8601TZ format writes time values in one of the following ISO 8601 extended time notations:

- *hh:mm:ss*+|–*hh:mm*

  Note: Use this form when *w* is large enough to support this time zone notation.

- *hh:mm:ss*Z

  Note: Use this form when *w* is not large enough to support the +|–*hh:mm* time zone notation.

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

+|–*hh:mm*

is an hour and minute signed offset from zero meridian time. Note that the offset must be +|–*hh:mm* (that is, + or – and five characters).

Restriction: The shorter form +|–*hh* is not supported.

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +02:00 indicates a two–hour time difference to the east of the zero meridian, and –06:00 indicates a six- hour time difference to the west of the zero meridian.

Z

indicates zero meridian (Greenwich, England) or +00:00 UTC time.

When SAS reads a UTC time by using the E8601TZ informat, and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 00:00:00 and 24:00:00. If the E8601TZ format attempts to format a time outside of this time range, the time is formatted with asterisks to indicate that the value is out of range.

Comparisons

For time values between 00:00:00 and 24:00:00, the the time value E8601TZ format adjusts the time value to be the time at the zero meridian and writes the time value in the international standard extended time notation. The E8601LZ format makes no adjustment to the time and writes time values in the international standard extended time notation, using a UTC time zone offset for the local SAS session.

Example

```
put etz e8601tz.;
```

Value of etz	Result
17024	04:43:44+00:00
85424	23:43:44+00:00

See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 14

FLOAT*w.d* Format

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

specifies the power of 10 by which to multiply the value. This argument is optional.

Default: 0

Range: 0-31

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 values that are 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 might 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)

Example

```
put x float4.;
```

Value of x	Result *
1	3F800000

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

FRACT*w*. Format

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.

Example

```
put x fract8.;
```

Value of x	Result
	`----+----1`
`0.6666666667`	`2/3`
`0.2784`	`174/625`

HEX*w*. Format

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

Category: Numeric

Alignment: left

See: "HEXw. Format: Windows" in *SAS Companion for Windows*
"HEXw. Format: UNIX" in *SAS Companion for UNIX Environments*
"HEXw. Format: z/OS" in *SAS Companion for z/OS*

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 characters. 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.

Example

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

Value of x	Result
	----+----1----+----2
35.4	00000023
88	00000058
2.33	00000002
-150	FFFFFF6A

HHMM*w.d* Format

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

specifies the number of digits to the right of the decimal point in the minutes value. The digits to the right of the decimal point specify a fraction of a minute. This argument is optional.

Default: 0

Range: 0–19

Requirement: must be less than *w*

Details

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

hh

is an integer.

Note: If *hh* is a single digit, HHMM*w.d* places a leading blank before the digit. For example, the HHMM*w.d.* format writes 9:00 instead of 09:00.

mm

is an integer between 00 and 59 that represents minutes.

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

The HHMM format uses asterisks to format values that are outside the time range 0–24 hours, such as datetime values.

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.

The HHMM*w.d* format writes a leading blank for a single-hour digit. The TOD*w.d* format writes a leading zero for a single-hour digit.

Example

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

SAS Statement	Result
	`----+----1`
`put time hhmm.;`	`13:00`
`put time hhmm8.2;`	`12:59.93`

In the first example, SAS rounds up the time value four seconds based on the value of seconds in the SAS time value. In the second example, by adding a decimal specification of 2 to the format shows that fifty-six seconds is 93% of a minute.

See Also

Formats:

- "HOURw.d Format" on page 105

- "MMSSw.d Format" on page 117
- "TIMEw.d Format" on page 157
- "TODw.d Format" on page 160

Functions:

- "HMS Function" in *SAS Functions and CALL Routines: Reference*
- "HOUR Function" in *SAS Functions and CALL Routines: Reference*
- "MINUTE Function" in *SAS Functions and CALL Routines: Reference*
- "SECOND Function" in *SAS Functions and CALL Routines: Reference*
- "TIME Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "TIMEw. Informat" on page 332

HOUR*w.d* Format

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

specifies the number of digits to the right of the decimal point in the hour value. Therefore, SAS prints decimal fractions of the hour. This argument is optional.

Range: 0-19

Requirement: must be less than *w*

Details

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

The HOUR format uses asterisks to format values that are outside the time range 0–24 hours, such as datetime values.

Example

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

SAS Statement	Result
	`----+----1`
`put time hour4.1;`	`11.5`

See Also

Formats:

- "HHMMw.d Format" on page 103
- "MMSSw.d Format" on page 117
- "TIMEw.d Format" on page 157
- "TODw.d Format" on page 160

Functions:

- "HMS Function" in *SAS Functions and CALL Routines: Reference*
- "HOUR Function" in *SAS Functions and CALL Routines: Reference*
- "MINUTE Function" in *SAS Functions and CALL Routines: Reference*
- "SECOND Function" in *SAS Functions and CALL Routines: Reference*
- "TIME Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "TIMEw. Informat" on page 332

IB*w.d* Format

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

Category: Numeric

Alignment: left

See: "IBw.d Format: UNIX" in *SAS Companion for UNIX Environments*
"IBw.d Format: Windows" in *SAS Companion for Windows*
"IBw.d Format: z/OS" in *SAS Companion for z/OS*

Syntax

IB*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 4

Range: 1–8

d

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

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 for Integer Binary Data on Big Endian and Little Endian Platforms" on page 7 .

Comparisons

The IB*w.d* and PIB*w.d* formats are used to write native format integers. (Native format enables 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 1.1 on page 8 .

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

Value of x	Result on Big Endian Platforms *	Result 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

Formats:

- "IBRw.d Format" on page 108

IBR*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

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 for Integer Binary Data on Big Endian and Little Endian Platforms" on page 7 .

Comparisons

- The IB*w.d* and PIB*w.d* formats are used to write native format integers. (Native format enables 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 that you are writing on.
- In Intel and DEC operating environments, the IB*w.d* and IBR*w.d* formats are equivalent.

To view the type of format to use with big endian and little endian integers, see Table 1.1 on page 8.

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

Value of x	Result*
	----+----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

Formats:

- "IBw.d Format" on page 106

IEEE*w.d* Format

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

Category: Numeric

Alignment: left

CAUTION: Large floating-point values and floating-point values that require precision might not be identical to the original SAS value when they are written to an IBM mainframe by using the IEEE format and read back into SAS using the IEE informat.

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

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.

Example

```
test1=put(x,ieee4.);
put test1 $hex8.;
test2=put(x,ieee5.);
put test2 $hex10.;
```

Value of x	Result*
1	3F800000
	3FF0000000

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

JULDAY*w*. Format

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).

Example

The example table uses the input values of 18993, which is the SAS date value that corresponds to January 1, 2012, and 19068, which is the SAS date value that corresponds to March 16, 2012.

Input Value	SAS Statement	Result
		`----+----1`
`18993`	`put date julday3.;`	`  1`
`19068`	`put date julday3.;`	` 76`

JULIAN*w*. Format

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*:

yy or *yyyy*
: is a two-digit 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.

Example

The example table uses the input value of 19114, which is the SAS date value that corresponds to May 1, 2012 (the 122nd day of the year).

SAS Statement	Result
	`----+----1`
`put date julian5.;`	`12122`
`put date julian7.;`	`2012122`

See Also

Functions:

- "DATEJUL Function" in *SAS Functions and CALL Routines: Reference*
- "JULDATE Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "JULIANw. Informat" on page 289

MDYAMPM*w.d* Format

Writes datetime values in the form *mm/dd/yy<yy> hh:mm* AM|PM. The year can be either two or four digits.

Category: Date and Time

Alignment: right

Note: The default time period is AM.

Syntax

MDYAMPM*w*.

Syntax Description

w

specifies the width of the output field.

Default: 19

Range: 8–40

Details

The MDYAMPM*w.d* format writes SAS datetime values in the following form:

mm/*dd*/*yy*<yy> *hh*:*mm*<AM | PM>:

mm

is an integer between 1 and 12 that represents the month.

dd

is an integer between 1 and 31 that represents the day of the month.

yy or *yyyy*

specifies a two-digit or four-digit integer that represents the year.

hh
is an integer between 00 and 23 that represents hours.

mm
is an integer between 00 and 59 that represents minutes.

AM | PM
specifies either the time period 00:01−12:00 noon (AM) or the time period 12:01−12:00 midnight (PM). The default is AM.

date and time separator characters
is one of several special characters, such as the slash (/), colon (:), or a blank character that SAS uses to separate date and time components.

Comparisons

The MDYAMPM*w.* format writes datetime values with separators in the form *mm/dd/yy<yy> hh:mm* AM | PM, and requires a space between the date and the time.

The DATETIME*w.d* format writes datetime values with separators in the form *ddmmmyy<yy>: hh:mm:ss.ss*.

Example

This example uses the input value of 1663343580, which is the SAS datetime value that corresponds to 3:53:00 PM on September 15, 2012.

SAS Statement	Result
`put dt mdyampm25.`	`9/15/2012 3:53 PM`

See Also

Formats:

- "DATETIMEw.d Format" on page 75

Informats:

- "MDYAMPMw.d Informat" on page 291

MMDDYY*w.* Format

Writes date values in the form *mmdd<yy>yy* or *mm/dd/<yy>yy*, where a forward slash is the separator and the year appears as either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

MMDDYY*w.*

Syntax Description

w
specifies the width of the output field.

Default: 8

Range: 2–10

Interaction: When *w* has a value of from 2 to 5, the date appears with as much of the month and the day as possible. When *w* is 7, the date appears as a two-digit year without slashes.

Details

The MMDDYY*w*. format writes SAS date values in one of the following forms:

mmdd<yy>yy

mm/dd/<yy>yy:

where

mm
is an integer that represents the month.

/
is the separator.

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

<yy>yy
is a two-digit or four-digit integer that represents the year.

Example

The following examples use the input value of 19291, which is the SAS date value that corresponds to October 25, 2012.

SAS Statement	Result
	`----+----1----+`
`put day mmddyy2.;`	`10`
`put day mmddyy3.;`	` 10`
`put day mmddyy4.;`	`1025`
`put day mmddyy5.;`	`10/25`
`put day mmddyy6.;`	`102512`
`put day mmddyy7.;`	` 102512`
`put day mmddyy8.;`	`10/25/12`
`put day mmddyy10.;`	`10/25/2012`

See Also

Formats:

- "DATEw. Format" on page 73
- "DDMMYYw. Format" on page 78
- "MMDDYYxw. Format" on page 115
- "YYMMDDw. Format" on page 181

Functions:

- "DAY Function" in *SAS Functions and CALL Routines: Reference*
- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267
- "DDMMYYw. Informat" on page 270
- "YYMMDDw. Informat" on page 347

MMDDYY*xw*. Format

Writes date values in the form *mmdd<yy>yy* or *mm-dd-<yy>yy*, where the *x* in the format name is a character that represents the special character which separates the month, day, and year. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

MMDDYY*xw*.

Syntax Description

x

identifies a separator or specifies that no separator appear between the month, the day, and the year. These are valid values for *x*:

B

separates with a blank.

C

separates with a colon.

D

separates with a hyphen.

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

Interactions:

When *w* has a value of from 2 to 5, the date appears with as much of the month and the day as possible. When *w* is 7, the date appears as a two-digit year without separators.

When *x* has a value of N, the width range changes to 2–8.

Details

The MMDDYY*xw*. format writes SAS date values in one of the following forms:

mmdd<yy>yy

mmxddx<yy>yy

where

mm
: is an integer that represents the month.

x
: is a specified separator.

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

<yy>yy
: is a two-digit or four-digit integer that represents the year.

Example

The following examples use the input value of 19127, which is the SAS date value that corresponds to May 14, 2012.

SAS Statement	Result
	`----+----1----+`
`put day mmddyyc5.;`	`05:14`
`put day mmddyyd8.;`	`05-14-12`
`put day mmddyyp10.;`	`05.14.2012`
`put day mmddyyn8.;`	`05142012`

See Also

Formats:

- "DATEw. Format" on page 73
- "DDMMYYxw. Format" on page 79
- "MMDDYYw. Format" on page 113
- "YYMMDDxw. Format" on page 183

Functions:

- "DAY Function" in *SAS Functions and CALL Routines: Reference*
- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "MMDDYYw. Informat" on page 292

MMSS*w.d* Format

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: Set *w* to a minimum of 5 to write a value that represents minutes and seconds.

d

specifies the number of digits to the right of the decimal point in the seconds value. Therefore, the SAS time value includes fractional seconds. This argument is optional.

Range: 0–19

Restriction: must be less than *w*

Details

The MMSS format uses asterisks to format values that are outside the time range 0–24 hours, such as datetime values.

Example

The example uses the input value of 4530.

SAS Statement	Result
	`----+----1`
`put time mmss.;`	`75:30`

See Also

Formats:

- "HHMMw.d Format" on page 103
- "TIMEw.d Format" on page 157

Functions:

- "HMS Function" in *SAS Functions and CALL Routines: Reference*
- "MINUTE Function" in *SAS Functions and CALL Routines: Reference*
- "SECOND Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "TIMEw. Informat" on page 332

MMYY*w*. Format

Writes date values in the form *mm*M<*yy*>*yy*, where M is the separator and the year appears as either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

MMYY*w*.

Syntax Description

w

specifies the width of the output field.

Default: 7

Range: 5–32

Interaction: When *w* has a value of 5 or 6, the date appears with only the last two digits of the year. When *w* is 7 or more, the date appears with a four-digit year.

Details

The MMYY*w*. format writes SAS date values in the form *mm*M<*yy*>*yy*, where

mm
: is an integer that represents the month.

M
: is the character separator.

<*yy*>*yy*
: is a two-digit or four-digit integer that represents the year.

Example

The following examples use the input value of 19291, which is the SAS date value that corresponds to October 25, 2012.

SAS Statement	Result
	`----+----1----+`
`put date mmyy5.;`	`10M12`
`put date mmyy6.;`	` 10M12`
`put date mmyy.;`	`10M2012`
`put date mmyy7.;`	`10M2012`
`put date mmyy10.;`	`   10M2012`

See Also

Formats:

- "MMYYxw. Format" on page 119
- "YYMMw. Format" on page 180

MMYY*xw*. Format

Writes date values in the form *mm*<*yy*>*yy* or *mm*-<*yy*>*yy*, where the *x* in the format name is a character that represents the special character that separates the month and the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

MMYY*xw*.

Syntax Description

x

identifies a separator or specifies that no separator appear between the month and the year. These are valid values for *x*:

C

separates with a colon.

D

separates with a hyphen.

N

indicates no separator.

P

separates with a period.

S

separates with a forward slash.

w

specifies the width of the output field.

Default: 7

Range: 5–32

Interactions:

When *x* is set to N, no separator is specified. The width range is then 4–32, and the default changes to 6.

When *x* has a value of C, D, P, or S and *w* has a value of 5 or 6, the date appears with only the last two digits of the year. When *w* is 7 or more, the date appears with a four-digit year.

When *x* has a value of *N* and *w* has a value of 4 or 5, the date appears with only the last two digits of the year. When *x* has a value of *N* and *w* is 6 or more, the date appears with a four-digit year.

Details

The MMYY*xw*. format writes SAS date values in one of the following forms:

mm<yy>yy

mmx<yy>yy

where

mm

is an integer that represents the month.

x

is a specified separator.

<yy>yy

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

Example

The following examples use the input value of 19127, which is the SAS date value that corresponds to May 14, 2012.

SAS Statement	Result
	`----+----1----+`
`put date mmyyc5.;`	`05:12`
`put date mmyyd.;`	`05-2012`
`put date mmyyn4.;`	`0512`
`put date mmyyp8.;`	` 05.2012`
`put date mmyys10.;`	`   05/2012`

See Also

Formats:

- "MMYYw. Format" on page 118
- "YYMMxw. Format" on page 185

MONNAME*w*. Format

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.

Example

The example table uses the input value of 19057, which is the SAS date value that corresponds to March 5, 2012.

SAS Statement	Result
	`----+----1`
`put date monname1.;`	`M`
`put date monname3.;`	`Mar`
`put date monname5.;`	`March`

See Also

Formats:

- "MONTHw. Format" on page 122

MONTH*w*. Format

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: 1–32

Tip: Use MONTH1. to obtain a hexadecimal value.

Details

The MONTH*w*. format writes the month (1 through 12) of the year from a SAS date value. If the month is a single digit, the MONTH*w*. format places a leading blank before the digit. For example, the MONTH*w*. format writes 4 instead of 04.

Example

The example table uses the input value of 19127, which is the SAS date value that corresponds to May 14, 2012.

SAS Statement	Result
	`----+----1`

SAS Statement	Result
`put date month.;`	5

See Also

Formats:

- "MONNAMEw. Format" on page 121

MONYY*w*. Format

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-digit or four-digit integer that represents the year.

Comparisons

The MONYY*w*. format and the DTMONYY*w*. format are similar in that they both write date values. The difference is that MONYY*w*. expects a SAS date value as input, and DTMONYY*w*. expects a datetime value.

Example

The example table uses the input value of 19127, which is the SAS date value that corresponds to May 14, 2012.

SAS Statement	Result
	`----+----1`
`put date monyy5.;`	`MAY12`
`put date monyy7.;`	`MAY2012`

See Also

Formats:

- "DTMONYYw. Format" on page 86
- "DDMMYYw. Format" on page 78
- "MMDDYYw. Format" on page 113
- "YYMMDDw. Format" on page 181

Functions:

- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informat:

- "MONYYw. Informat" on page 294

NEGPAREN*w.d* Format

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

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Default: 0

Range: 0–31

Details

The NEGPAREN*w.d* format attempts to right-align output values. If the input value is negative, NEGPAREN*w.d* displays the output by enclosing the value in parentheses, if the field that you specify is wide enough. Otherwise, it uses a minus sign to represent the negative value. If the input value is nonnegative, NEGPAREN*w.d* displays the value with a leading and trailing blank to ensure proper column alignment. It reserves the last column for a close parenthesis even when the value is positive.

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.

Example

```
put @1 sales negparen8.;
```

Value of `sales`	Result
	`----+----1----+`
`100`	`     100`
`1000`	`   1,000`
`-200`	`    (200)`
`-2000`	`  (2,000)`

NUMX*w.d* Format

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

specifies the number of digits to the right of the decimal point (comma) in the numeric value. This argument is optional.

Default: 0

Range: 0–31

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.

Example: Examples

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

Value of x	Result
`----+----1----+`	
`896.48`	`896,48`
`64.89`	`64,89`
`3064.10`	`3064,10`

See Also

Formats:

- "w.d Format" on page 164

Informats:

- "NUMXw.d Informat" on page 296

OCTAL*w.* Format

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.

Comparisons

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

Example

```
put x octal6.;
```

Value of x	Result
	----+----1
3592	007010

PD*w.d* Format

Writes data in packed decimal format.

Category: Numeric

Alignment: left

See: "PDw.d Format: UNIX" in *SAS Companion for UNIX Environments*
"PDw.d Format: Windows" in *SAS Companion for Windows*
"PDw.d Format: z/OS" in *SAS Companion for z/OS*

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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 type of operating environment that you use to run SAS.

The PD*w.d* format writes missing numerical data as –0. When the PD*w.d* informat reads a –0, it stores it as 0.

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

Example

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

Value of x	Result *
	----+----1
128	00000128

* 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.* Format

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: 3-16

Details

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

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.

Example

SAS Statement	Result
	`----+----1`

`date = '17mar2012'd;`
`juldate = put(date,pdjulg4.);`
`put juldate $hex8.;`
	`2012077F`

See Also

Formats:

- "PDJULIw. Format" on page 130
- "JULIANw. Format" on page 111
- "JULDAYw. Format" on page 110

Functions:

- "JULDATE Function" in *SAS Functions and CALL Routines: Reference*
- "DATEJUL Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "PDJULIw. Informat" on page 301

- "PDJULGw. Informat" on page 300
- "JULIANw. Informat" on page 289

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

PDJULI*w*. Format

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: 3-16

Details

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

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.

Example

SAS Statement	Result
	----+----1

SAS Statement	Result

`date = '17mar2012'd;`
`juldate = put(date,pdjuli4.);`
`put juldate $hex8.;`
	`0112077F`

`date = '31dec2013'd;`
`juldate = put(date,pdjuli4.);`
`put juldate $hex8.;`
	`0113365F`

See Also

Formats:

- "PDJULGw. Format" on page 128
- "JULIANw. Format" on page 111
- "JULDAYw. Format" on page 110

Functions:

- "DATEJUL Function" in *SAS Functions and CALL Routines: Reference*
- "JULDATE Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "PDJULGw. Informat" on page 300
- "PDJULIw. Informat" on page 301
- "JULIANw. Informat" on page 289

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

PERCENT*w.d* Format

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

Tip: The width of the output field must account for the percent sign (%) and parentheses for negative numbers, whether the number is negative or positive.

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Range: 0–31

Requirement: must be less than *w*

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.

Example

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

Value of x	Result
	`----+----1----+----2`
`0.1`	`       10%`
`1.2`	`      120%`
`-0.05`	`(      5%)`

See Also

Formats:

- "PERCENTNw.d Format" on page 132

PERCENTN*w.d* Format

Produces percentages, using a minus sign for negative values.

Category: Numeric

Alignment: right

Syntax

PERCENTN*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 6

Range: 4–32

Tip: The width of the output field must account for the minus sign (–), the percent sign (%), and a trailing blank, whether the number is negative or positive.

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Range: 0–31

Requirement: must be less than *w*

Details

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

Comparisons

The PERCENTN*w.d* format produces percents by using a minus sign instead of parentheses for negative values. The PERCENT*w.d* format produces percents by using parentheses for negative values.

Example

```
put x percentn10.;
```

Value of x	Result
-0.1	-10%
.2	20%
.8	80%
-0.05	-5%
-6.3	-630%

See Also

Format:

- "PERCENTw.d Format" on page 131

PIB*w.d* Format

Writes positive integer binary (fixed-point) values.

Category: Numeric

Alignment: left

See: "PIBw.d Format: UNIX" in *SAS Companion for UNIX Environments*
"PIBw.d Format: Windows" in *SAS Companion for Windows*

Syntax

PIB*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 1

Range: 1–8

d

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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 for Integer Binary Data on Big Endian and Little Endian Platforms" on page 7 .

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. A value that corresponds to the binary equivalent of the contents of a byte 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 enables 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 1.1 on page 8 .

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

Value of `x`	Result *
	`----+----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:

- "PIBRw.d Format" on page 135

PIBR*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

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 for Integer Binary Data on Big Endian and Little Endian Platforms" on page 7 .

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. A value that corresponds to the binary equivalent of the contents of a byte 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 enables 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 1.1 on page 8 .

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

Value of x	Result *
	----+----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

Informats:

- "PIBw.d Informat" on page 304

PK*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

Requirement: must be less than *w*

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.

Example

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

Value of x	Result *
	`----+----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* Format

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

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Default: the minimum of 4 and *w*–2

Range: 1–30

Restriction: must be less than *w*

Comparisons

The PVALUE*w.d* format follows the rules for the *w.d* format, except in the following conditions:

- if the value x is such that $0 <= x < 10^{-d}$, x prints as "<.0...01" with *d*-1 zeros
- missing values print as "." unless you specify a different character by using the MISSING= system option

Example

```
put x pvalue6.4;
```

Value of x	Result
	`----+----1`
`.05`	`0.0500`
`0.000001`	`<.0001`
`0`	`<.0001`
`.0123456`	`0.0123`

QTR*w*. Format

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

Example

The example table uses the input value of 19057, which is the SAS date value that corresponds to March 5, 2012.

SAS Statement	Result
	`----+----1`
`put date qtr.;`	`1`

See Also

Formats:

- "QTRRw. Format" on page 139

QTRR*w*. Format

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

Example

The example table uses the input value of 19251, which is the SAS date value that corresponds to September 15, 2012.

SAS Statement	Result
	`----+----1`
`put date qtrr.;`	`III`

See Also

Formats:

- "QTRw. Format" on page 138

RB*w.d* Format

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

Category: Numeric

Alignment: left

See: "RBw.d Format: UNIX" in *SAS Companion for UNIX Environments*
"RBw.d Format: Windows" in *SAS Companion for Windows*
"RBw.d Format: z/OS" in *SAS Companion for z/OS*

Syntax

RB*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 4

Range: 2–8

d

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

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 type 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

Example

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

Value of x	Result *
	----+---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*. Format

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.

Example

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

Value of `year`	Result
2012	MMXII

S370FF*w.d* Format

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

specifies the power of 10 by which to divide the value. This argument is optional.

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.

Example

```
y=put(x,s370ff5.);
put y $hex10.;
```

Value of x	Result *
	----+----1
12345	F1F2F3F4F5

* The result is the hexadecimal representation for the integer.

See Also

Formats:

- “$EBCDICw. Format” on page 37
- “w.d Format” on page 164

S370FIB*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

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 for Integer Binary Data on Big Endian and Little Endian Platforms" on page 7 .

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 1.1 on page 8.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

Value of x	Result *
	----+----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:

- "S370FIBUw.d Format" on page 144
- "S370FPIBw.d Format" on page 148

S370FIBU*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

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 for Integer Binary Data on Big Endian and Little Endian Platforms" on page 7 .

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 1.1 on page 8.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

Value of x	Result *
245	F5

Value of x	Result *
-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

- "S370FIBw.d Format" on page 143
- "S370FPIBw.d Format" on page 148

S370FPD*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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.

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

Example

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

Value of x	Result *
	----+----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* Format

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

specifies to multiply the number by 10^{d}. This argument is optional.

Default: 0

Range: 0–31

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.

Example

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

Value of x	Result *
123	123F
-123	123F

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

S370FPIB*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

Details

Positive integer binary values are the same as integer binary values, except that all values are treated as positive. S370FPIB*w.d* 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 for Integer Binary Data on Big Endian and Little Endian Platforms" on page 7 .

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 1.1 on page 8.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

Value of x	Result *
	----+----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:

- "S370FIBw.d Format" on page 143
- "S370FIBUw.d Format" on page 144

S370FRB*w.d* Format

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: 4

Range: 2–8

d

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–10

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
IBM 370 assembler	E	D
C	float	double

Example

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

Value of x	Result *
128	42800000

Value of x	Result *
-123	C2800000

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

S370FZD*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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

Example

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

Value of x	Result *
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 characters correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZDL*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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.

Example

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

Value of x	Result *
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 characters correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZDS*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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.

Example

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

Value of x	Result *
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 characters correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZDT*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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.

Example

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

Value of x	Result *
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 characters correspond to one byte of binary data, and each byte corresponds to one column of the output field.

S370FZDU*w.d* Format

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

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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.

Example

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

Value of `x`	Result *
123	F1F2F3
-123	F1F2F3

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each pair of hexadecimal characters (such as F1) corresponds to one byte of binary data, and each byte corresponds to one column of the output field.

SSN*w*. Format

Writes Social Security numbers.

Category: Numeric

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 hyphens 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.

Example

```
put id ssn.;
```

Value of `id`	Result
	----+----1----+
263878439	263-87-8439

TIME*w.d* Format

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

Tip: 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: eight spaces to the left of the decimal point, one space for the decimal point itself, and three spaces for the decimal fraction of seconds.

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–19

Requirement: must be less than *w*

Details

The TIME*w.d* format writes SAS time values in the form *hh:mm:ss.ss*:

hh

is an integer.

Note: If *hh* is a single digit, TIME*w.d* places a leading blank before the digit. For example, the TIME*w.d.* format writes 9:00 instead of 09:00.

mm

is an integer between 00 and 59 that represents minutes.

ss.ss

is the number of seconds between 00 and 59, with the fraction of a second following the decimal point.

Comparisons

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

The TIME*w.d* format writes a leading blank for a single-hour digit. The TOD*w.d* format writes a leading zero for a single-hour digit.

Examples

Example 1

This example uses the input value of 59083, which is the SAS time value that corresponds to 4:24:43 p.m.

SAS Statement	Result
	`----+----1`
`put begin time.;`	`16:24:43`

Example 2

This example uses the input value of 32083, which is the SAS time value that corresponds to 8:54:43 a.m.

SAS Statement	Result
	`----+----1`
`put begin time.;`	` 8:54:43`

See Also

Formats:

- “HHMMw.d Format” on page 103
- “HOURw.d Format” on page 105
- “MMSSw.d Format” on page 117
- “TODw.d Format” on page 160

Functions:

- “HOUR Function” in *SAS Functions and CALL Routines: Reference*
- “MINUTE Function” in *SAS Functions and CALL Routines: Reference*
- “SECOND Function” in *SAS Functions and CALL Routines: Reference*
- “TIME Function” in *SAS Functions and CALL Routines: Reference*

Informats:

- “TIMEw. Informat” on page 332

TIMEAMPM*w.d* Format

Writes time and datetime 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

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–19

Requirement: must be less than *w*

Details

The TIMEAMPM*w.d* format writes SAS time values and SAS datetime 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.

Example

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

SAS Statement	Result
	`----+----1----+`

SAS Statement	Result
`put begin timeampm3.;`	`PM`
`put begin timeampm5.;`	`4 PM`
`put begin timeampm7.;`	`4:24 PM`
`put begin timeampm11.;`	`4:24:43 PM`

See Also

Formats:

- "TIMEw.d Format" on page 157

TOD*w.d* Format

Writes SAS time values and the time portion of SAS 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

Tip: SAS writes a zero for a zero hour if the specified width is sufficient. For example, 02:30 or 00:30.

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–19

Requirement: must be less than *w*

Details

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

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 TOD*w.d* format writes a leading zero for a single-hour digit. The TIME*w.d* format and the HHMM*w.d* format write a leading blank for a single-hour digit.

Examples

Example 1

In this example, the SAS datetime value 1661437223 corresponds to August 24, 2012 at 2:20:23 p.m.

SAS Statement	Result
	`----+----1`
`begin = '1:30't;` `put begin tod5.;`	`01:30`
`begin = 1661437223;` `put begin tod9.;`	`14:20:23`

Example 2

In this example, the SAS time value 32083 corresponds to 8:54:43 a.m.

SAS Statement	Result
	`----+----1`
`begin = 32083;` `put begin tod9.;`	`08:54:43`

See Also

Formats:

- “HHMMw.d Format” on page 103
- “TIMEw.d Format” on page 157
- “TIMEAMPMw.d Format” on page 158

Functions:

- “TIMEPART Function” in *SAS Functions and CALL Routines: Reference*

Informats:

• "TIMEw. Informat" on page 332

VAXRB*w.d* Format

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

Category: Numeric

Alignment: right

Syntax

VAXRB*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 2-8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–31

Details

Use the VAXRB*w.d* format to write data in native VAX or VMS floating-point notation.

Comparisons

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

Example

```
x=1;
y=put(x,vaxrb8.);
put y=$hex16.;
```

Value of x	Result *
----+----1	
1	8040000000000000

* The result is the hexadecimal representation for the integer.

VMSZN*w.d* Format

Generates VMS and MicroFocus COBOL zoned numeric data.

Category: Numeric

Alignment: left

Syntax

VMSZN*w.d*

Required Arguments

w

specifies the width of the output field

Default: 1

Range: 1–32

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Details

The VMSZN*w.d* format is similar to the ZD*w.d* format. Both generate a string of ASCII digits, and the last digit is a special character that denotes the magnitude of the last digit and the sign of the entire number. The difference between these formats is in the special character that is used for the last digit. The following table shows the special characters that are used by the VMSZN*w.d* format.

Desired Digit	Special Character	Desired Digit	Special Character
0	0	–0	p
1	1	–1	q
2	2	–2	r
3	3	–3	s
4	4	–4	t
5	5	–5	u
6	6	–6	v
7	7	–7	w
8	8	–8	x

Desired Digit	Special Character	Desired Digit	Special Character
9	9	–9	y

Data formatted using the VMSZN*w.d* format are ASCII strings.

If the value to be formatted is too large to fit in a field of the specified width, then the VMSZN*w.d* format does the following:

- For positive values, it sets the output to the largest positive number that fits in the given width.
- For negative values, it sets the output to the negative number of greatest magnitude that fits in the given width.

Example

SAS Statements	Result
	`----+----1`
`x=1234;` `put x vmszn4.;`	`1234`
`x=1234;` `put x vmszn5.1;`	`12340`
`x=1234;` `put x vmszn6.2;`	`123400`
`-1234;` `put x vmszn5.;`	`0123t`

See Also

Formats:

- "ZDw.d Format" on page 194

Informats:

- "VMSZNw.d Informat" on page 337

w.d Format

Writes standard numeric data one digit per byte.

Category: Numeric

Alignment: right

Alias: F*w.d*

See: "w.d Format: z/OS" in *SAS Companion for z/OS*

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

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Range: 0–31

Requirement: must be less than *w*

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 might 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.

Example

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

Value of x	Result
	`----+----1----+`
`23.45`	`      23.450`

WEEKDATE*w*. Format

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*):

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

yy or *yyyy*
: is a two-digit 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 WEEKDATX*w*. prints *dd* before the month's name.

Example

The example table uses the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

SAS Statement	Result
	`----+----1----+----2`
`put date weekdate3.;`	`Thu`
`put date weekdate9.;`	` Thursday`
`put date weekdate15.;`	`Thu, Jun 14, 12`
`put date weekdate17.;`	`Thu, Jun 14, 2012`

See Also

Formats:

- “DATEw. Format” on page 73
- “DDMMYYw. Format” on page 78

- "MMDDYYw. Format" on page 113
- "TODw.d Format" on page 160
- "WEEKDATXw. Format" on page 167
- "YYMMDDw. Format" on page 181

Functions:

- "JULDATE Function" in *SAS Functions and CALL Routines: Reference*
- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "WEEKDAY Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267
- "DDMMYYw. Informat" on page 270
- "MMDDYYw. Informat" on page 292
- "YYMMDDw. Informat" on page 347

WEEKDATX*w*. Format

Writes date values as the day of the 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*):

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

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

If *w* is too small to write the complete day of the week and month, then 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.

The WEEKDATX*w*. format is the same as the DTWKDATX*w*. format, except that DTWKDATX*w*. expects a datetime value as input.

Example

The example table uses the input value of 19046, which is the SAS date value that corresponds to February 23, 2012.

SAS Statement	Result
	`----+----1----+----2----+----3`
`put date weekdatx.;`	`    Thursday, 23 February 2012`

See Also

Formats:

- "DTWKDATXw. Format" on page 87
- "DATEw. Format" on page 73
- "DDMMYYw. Format" on page 78
- "MMDDYYw. Format" on page 113
- "TODw.d Format" on page 160
- "WEEKDATEw. Format" on page 165
- "YYMMDDw. Format" on page 181

Functions:

- "JULDATE Function" in *SAS Functions and CALL Routines: Reference*
- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "WEEKDAY Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267
- "DDMMYYw. Informat" on page 270
- "MMDDYYw. Informat" on page 292
- "YYMMDDw. Informat" on page 347

WEEKDAY*w*. Format

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).

Example

The example table uses the input value of 19025, which is the SAS date value that corresponds to February 2, 2012.

SAS Statement	Result
	`----+----1`
`put date weekday.;`	`5`

See Also

Formats:

- "DOWNAMEw. Format" on page 84

WEEKU*w*. Format

Writes a week number in decimal format by using the U algorithm.

Category: Date and Time

Alignment: left

Syntax

WEEKU*w*.

Syntax Description

w

specifies the width of the output field.

Default: 11

Range: 3–200

Details

The WEEKU*w*. format writes a week-number format. The WEEKU*w*. format writes the various formats depending on the specified width. Algorithm U calculates the SAS date value by using the number of the week within the year (Sunday is considered the first day of the week). The number-of-the-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

Refer to the following table for widths, formats, and examples:

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	12W01
7-8	yyWwwdd	12W0101
9-10	yyyyWwwdd	2012W0101
11-200	yyyy-Www-dd	2012-W01-01

Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53, with weeks beginning on a Monday and week 1 of the year including both January 4th and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. The WEEKW*w*. format writes the week number of the year as a decimal number in the range 00–53, with Monday as the first day of week 1. The WEEKU*w*. format writes the week number of the year (Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading zero.

Example

```
sasdate = '31JAN2012'd;
```

Statements	Result
	`----+----1----+`

Statements	Result

`v=put(sasdate,weeku3.);`
`w=put(sasdate,weeku5.);`
`x=put(sasdate,weeku7.);`
`y=put(sasdate,weeku9.);`
`z=put(sasdate,weeku11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	
	`W05`
	`12W05`
	`12W0503`
	`2012W0503`
	`2012-W05-03`

See Also

Formats:

- "WEEKVw. Format" on page 171
- "WEEKWw. Format" on page 173

Functions:

- "WEEK Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "WEEKUw. Informat" on page 340
- "WEEKVw. Informat" on page 341
- "WEEKWw. Informat" on page 343

WEEKV*w*. Format

Writes a week number in decimal format by using the V algorithm.

Category: Date and Time

Alignment: left

Syntax

WEEKV*w*.

Syntax Description

w

specifies the width of the output field.

Default: 11

Range: 3–200

Details

The WEEKV*w*. format writes the various formats depending on the specified width. Algorithm V calculates the SAS date value, with the number-of-the-week value represented as a decimal number in the range 01–53, with a leading zero and maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4th and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. For example, the fifth week of the year would be represented as 06.

Refer to the following table for widths, formats, and examples:

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	12W01
7-8	yyWwwdd	12W0101
9-10	yyyyWwwdd	2012W0101
11-200	yyyy-Www-dd	2012-W01-01

Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53, with weeks beginning on a Monday and week 1 of the year including both January 4th and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. The WEEKW*w*. format writes the week number of the year as a decimal number in the range 00–53, with Monday as the first day of week 1. The WEEKU*w*. format writes the week number of the year (Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading zero.

Example

```
sasdate='31JAN2012'd;
```

Statements	Result
	`----+----1----+`

`v=put(sasdate,weekv3.);`
`w=put(sasdate,weekv5.);`
`x=put(sasdate,weekv7.);`
`y=put(sasdate,weekv9.);`
`z=put(sasdate,weekv11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	
	`W05`
	`12W01`
	`12W0502`
	`2012W0502`
	`2012-W05-02`

See Also

Formats:

- "WEEKUw. Format" on page 169
- "WEEKWw. Format" on page 173

Functions:

- "WEEK Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "WEEKUw. Informat" on page 340
- "WEEKVw. Informat" on page 341
- "WEEKWw. Informat" on page 343

WEEKW*w*. Format

Writes a week number in decimal format by using the W algorithm.

Category: Date and Time

Alignment: left

Syntax

WEEKW*w*.

Syntax Description

w

specifies the width of the output field.

Default: 11

Range: 3–200

Details

The WEEKW*w*. format writes the various formats depending on the specified width. Algorithm W calculates the SAS date value using the number of the week within the year (Monday is considered the first day of the week). The number-of-the-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

Refer to the following table for widths, formats, and examples:

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	12W01

Widths	Formats	Examples
7-8	yyWwwdd	12W0101
9-10	yyyyWwwdd	2012W0101
11-200	yyyy-Www-dd	2012-W01-01

Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53. Weeks beginning on a Monday and on week 1 of the year include both January 4th and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. The WEEKW*w*. format writes the week number of the year as a decimal number in the range 00–53, with Monday as the first day of week 1. The WEEKU*w*. format writes the week number of the year (Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading zero.

Example

```
sasdate = '31JAN2012'd;
```

Statements	Result
	`----+----1----+`

`v=put(sasdate,weekw3.);`
`w=put(sasdate,weekw5.);`
`x=put(sasdate,weekw7.);`
`y=put(sasdate,weekw9.);`
`z=put(sasdate,weekw11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	
	`W05`
	`12W05`
	`12W0502`
	`2012W0502`
	`2012-W05-02`

See Also

Formats:

- “WEEKUw. Format” on page 169
- “WEEKVw. Format” on page 171

Functions:

- “WEEK Function” in *SAS Functions and CALL Routines: Reference*

Informats:

- “WEEKUw. Informat” on page 340

- "WEEKVw. Informat" on page 341
- "WEEKWw. Informat" on page 343

WORDDATE*w*. Format

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*:

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.

Example

The example table uses the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

SAS Statement	Result
`put term worddate3.;`	`Jun`
`put term worddate9.;`	`----+----1----+----2` `     June`
`put term worddate12.;`	`Jun 14, 2012`

SAS Statement	Result
`put term worddate20.;`	`----+----1----+----2` `June 14, 2012`

See Also

Formats:

- "WORDDATXw. Format" on page 176

WORDDATX*w*. Format

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*:

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.

Example

The example table uses the input value of 19057, which is the SAS date value that corresponds to March 5, 2012.

SAS Statement	Result
	`----+----1----+----2`
`put term worddatx.;`	`05 March 2012`

See Also

Formats:

- "WORDDATEw. Format" on page 175

WORDF*w*. Format

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.

Example

```
put price wordf15.;
```

Value of `price`	Result
	`----+----1----+`
`2.5`	`two and 50/100`

See Also

Formats:

- "WORDSw. Format" on page 178

WORDS*w*. Format

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.

Example

```
put price words23.;
```

Value of `price`	Result
	`----+----1----+----2----+`
`2.1`	`two and ten hundredths`

See Also

Formats:

- "WORDFw. Format" on page 177

YEAR*w*. Format

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.

Details

The YEAR*w*. format is similar to the DTYEAR*w*. format in that they both write date values. The difference is that YEAR*w*. expects a SAS date value as input, and DTYEAR*w*. expects a datetime value.

Example

The example table uses the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

SAS Statement	Result
	`----+----1`
`put date year2.;`	`12`

SAS Statement	Result
`put date year4.;`	`2012`

See Also

Formats:

- "DTYEARw. Format" on page 88

YYMM*w*. Format

Writes date values in the form <*yy*>*yy*M*mm*, where M is a character separator to indicate that the month number follows the M and the year appears as either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

YYMM*w*.

Syntax Description

w

specifies the width of the output field.

Default: 7

Range: 5–32

Interaction: When *w* has a value of 5 or 6, the date appears with only the last two digits of the year. When *w* is 7 or more, the date appears with a four-digit year.

Details

The YYMM*w*. format writes SAS date values in the form <*yy*>*yy*M*mm*:

<*yy*>*yy*
: is a two-digit or four-digit integer that represents the year.

M
: is the character separator to indicate that the number of the month follows.

mm
: is an integer that represents the month.

Example

The following examples use the input value of 19291, which is the SAS date value that corresponds to October 25, 2012.

SAS Statement	Result
	`----+----1----+`
`put date yymm5.;`	`12M10`
`put date yymm6.;`	` 12M10`
`put date yymm.;`	`2012M10`
`put date yymm7.;`	`2012M10`
`put date yymm10.;`	`----+----1----+` `   2012M10`

See Also

Formats:

- "MMYYw. Format" on page 118
- "YYMMxw. Format" on page 185

YYMMDD*w*. Format

Writes date values in the form *yymmdd* or *<yy>yy-mm-dd*, where a hyphen is the separator and the year appears as either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

YYMMDD*w*.

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 2–10

Interaction: When *w* has a value of from 2 to 5, the date appears with as much of the year and the month as possible. When *w* is 7, the date appears as a two-digit year without hyphens.

Details

The YYMMDD*w*. format writes SAS date values in one of the following forms:

yymmdd

<yy>yy–mm–dd

where

<yy>yy
: is a two-digit or four-digit integer that represents the year.

_
: is the separator.

mm
: is an integer that represents the month.

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

To format a date that has a four-digit year and no separators, use the YYMMDD*x*. format.

Example

The following examples use the input value of 19086, which is the SAS date value that corresponds to April 3, 2012.

SAS Statement	Result
	`----+----1----+`
`put day yymmdd2.;`	`12`
`put day yymmdd3.;`	` 12`
`put day yymmdd4.;`	`1204`
`put day yymmdd5.;`	`12-04`
`put day yymmdd6.;`	`120403`
`put day yymmdd7.;`	` 120403`
`put day yymmdd8.;`	`12-04-03`
`put day yymmdd10.;`	`2012-04-03`

See Also

Formats:

- “DATEw. Format” on page 73
- “DDMMYYw. Format” on page 78
- “MMDDYYw. Format” on page 113
- “YYMMDDxw. Format” on page 183

Functions:

- “DAY Function” in *SAS Functions and CALL Routines: Reference*

- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267
- "DDMMYYw. Informat" on page 270
- "MMDDYYw. Informat" on page 292

YYMMDD*xw*. Format

Writes date values in the form *yymmdd* or *<yy>yy-mm-dd*, where the *x* in the format name is a character that represents the special character which separates the year, month, and day. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

YYMMDD*xw*.

Syntax Description

x

identifies a separator or specifies that no separator appear between the year, the month, and the day. These are the valid values for *x*:

B

separates with a blank.

C

separates with a colon.

D

separates with a hyphen.

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

Interactions:

When *w* has a value of from 2 to 5, the date appears with as much of the year and the month. When *w* is 7, the date appears as a two-digit year without separators.

When *x* has a value of N, the width range is 2–8.

Details

The YYMMDD*xw*. format writes SAS date values in one of the following forms:

yymmdd

<yy>yyxmmxdd

where

<yy>yy
: is a two-digit or four-digit integer that represents the year.

x
: is a specified separator.

mm
: is an integer that represents the month.

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

Example

The following examples use the input value of 19127, which is the SAS date value that corresponds to May 14, 2012.

SAS Statement	Result
	`----+----1----+`
`put day yymmddc5.;`	`12:05`
`put day yymmddd8.;`	`12-05-14`
`put day yymmddp10.;`	`2012.05.14`
`put day yymmddn8.;`	`20120514`

See Also

Formats:

- “DATEw. Format” on page 73
- “DDMMYYxw. Format” on page 79
- “MMDDYYxw. Format” on page 115
- “YYMMDDw. Format” on page 181

Functions:

- “DAY Function” in *SAS Functions and CALL Routines: Reference*

- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "YYMMDDw. Informat" on page 347

YYMM*xw*. Format

Writes date values in the form <*yy*>*yymm* or <*yy*>*yy-mm*. The *x* in the format name represents the special character that separates the year and the month. This special character can be a hyphen (-), period (.), slash (/), colon (:), or no separator. The year can be either two or four digits.

Syntax

YYMM*xw*.

Syntax Description

x

identifies a separator or specifies that no separator appear between the year and the month. These are valid values for *x*:

C

separates with a colon.

D

separates with a hyphen.

N

indicates no separator.

P

separates with a period.

S

separates with a forward slash.

w

specifies the width of the output field.

Default: 7

Range: 5–32

Interactions:

When *x* is set to *N*, no separator is specified. The width range is then 4–32, and the default changes to 6.

When *x* has a value of C, D, P, or S and *w* has a value of 5 or 6, the date appears with only the last two digits of the year. When *w* is 7 or more, the date appears with a four-digit year.

When *x* has a value of *N* and *w* has a value of 4 or 5, the date appears with only the last two digits of the year. When *x* has a value of *N* and *w* is 6 or more, the date appears with a four-digit year.

Details

The YYMM*xw*. format writes SAS date values in one of the following forms:

<yy>yymm

*<yy>yy*X*mm*

where

<yy>yy
: is a two-digit or four-digit integer that represents the year.

x
: is a specified separator.

mm
: is an integer that represents the month.

Example

The following examples use the input value of 19127, which is the SAS date value that corresponds to May 14, 2012.

SAS Statement	Result
	`----+----1----+`
`put date yymmc5.;`	`12:05`
`put date yymmd.;`	`2012-05`
`put date yymmn4.;`	`1205`
`put date yymmp8.;`	` 2012.05`
`put date yymms10.;`	`   2012/05`

See Also

Formats:

- "MMYYxw. Format" on page 119
- "YYMMw. Format" on page 180

YYMON*w*. Format

Writes date values in the form *yymmm* or *yyyymmm*.

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 writes SAS date values in the form *<yy>yymmm*:

<yy>yy
: is a two-digit or four-digit integer that represents the year.

mmm
: is the name of the month, abbreviated to three characters.

Example

The example table uses the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

SAS Statement	Result
	`----+----1`
`put date yymon6.;`	` 02JUN`
`put date yymon7.;`	`2012JUN`

See Also

Formats:

- “MMYYw. Format” on page 118

YYQ*w*. Format

Writes date values in the form *<yy>yy*Q*q*, where Q is the separator, the year appears as either 2 or 4 digits, and *q* is the quarter of the year.

Category: Date and Time

Alignment: right

Syntax

YYQ*w*.

Syntax Description

w

specifies the width of the output field.

Default: 6

Range: 4–32

Interaction: When *w* has a value of 4 or 5, the date appears with only the last two digits of the year. When *w* is 6 or more, the date appears with a four-digit year.

Details

The YYQ*w*. format writes SAS date values in the form *<yy>yy*Q*q*:

<yy>yy
: is a two-digit or four-digit integer that represents the year.

Q
: is the character separator.

q
: is an integer (1,2,3, or 4) that represents the quarter of the year.

Example

The following examples use the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

SAS Statements	Result
	`----+----1----+`
`put date yyq4.;`	`12Q2`
`put date yyq5.;`	` 12Q2`
`put date yyq.;`	`2012Q2`
`put date yyq6.;`	`2012Q2`
`put date yyq10.;`	`    2012Q2`

See Also

Formats:

- "YYQxw. Format" on page 188
- "YYQRw. Format" on page 190

YYQ*xw*. Format

Writes date values in the form *<yy>yyq* or *<yy>yy-q*, where the *x* in the format name is a character that represents the special character that separates the year and the quarter or the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

YYQ*xw*.

Syntax Description

x

identifies a separator or specifies that no separator appear between the year and the quarter. Valid values for *x* are:

C

separates with a colon

D

separates with a hyphen

N

indicates no separator

P

separates with a period

S

separates with a forward slash.

w

specifies the width of the output field.

Default: 6

Range: 4–32

Interactions:

When *x* is set to *N*, no separator is specified. The width range is then 3–32, and the default changes to 5.

When *w* has a value of 4 or 5, the date appears with only the last two digits of the year. When *w* is 6 or more, the date appears with a four-digit year.

When *x* has a value of *N* and *w* has a value of 3 or 4, the date appears with only the last two digits of the year. When *x* has a value of *N* and *w* is 5 or more, the date appears with a four-digit year.

Details

The YYQ*xw*. format writes SAS date values in one of the following forms:

<yy>yyq

<yy>yyxq

where

<yy>yy

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

x

is a specified separator.

q

is an integer (1,2,3, or 4) that represents the quarter of the year.

Example

The following examples use the input value of 19188, which is the SAS date value that corresponds to July 14, 2012.

SAS Statement	Result
	`----+----1----+`
`put date yyqc4.;`	`12:3`
`put date yyqd.;`	`2012-3`
`put date yyqn3.;`	`123`
`put date yyqp6.;`	`2012.3`
`put date yyqs8.;`	`  2012/3`

See Also

Formats:

- "YYQw. Format" on page 187
- "YYQRxw. Format" on page 191

YYQR*w*. Format

Writes date values in the form <*yy*>*yy*Q*qr*, where Q is the separator, the year appears as either 2 or 4 digits, and *qr* is the quarter of the year expressed in roman numerals.

Category: Date and Time

Alignment: right

Syntax

YYQR*w*.

Syntax Description

w

specifies the width of the output field.

Default: 8

Range: 6–32

Interaction: When the value of *w* is too small to write a four-digit year, the date appears with only the last two digits of the year.

Details

The YYQR*w*. format writes SAS date values in the form *<yy>yy*Q*qr*:

<yy>yy
: is a two-digit or four-digit integer that represents the year.

Q
: is the character separator.

qr
: is a roman numeral (I, II, III, or IV) that represents the quarter of the year.

Example

The following examples use the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

SAS Statement	Result
	`----+----1----+`
`put date yyqr6.;`	` 12QII`
`put date yyqr7.;`	`2012QII`
`put date yyqr.;`	` 2012QII`
`put date yyqr8.;`	` 2012QII`
`put date yyqr10.;`	`   2012QII`

See Also

Formats:

- "YYQw. Format" on page 187
- "YYQRxw. Format" on page 191

YYQR*xw*. Format

Writes date values in the form *<yy>yyqr* or *<yy>yy-qr*, where the *x* in the format name is a character that represents the special character that separates the year and the quarter or the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits and *qr* is the quarter of the year expressed in roman numerals.

Category: Date and Time

Alignment: right

Syntax

YYQR*xw*.

Syntax Description

x
: identifies a separator or specifies that no separator appear between the year and the quarter. These are valid values for *x*:

  C
  : separates with a colon.

  D
  : separates with a hyphen.

  N
  : indicates no separator.

  P
  : separates with a period.

  S
  : separates with a forward slash.

w
: specifies the width of the output field.

  Default: 8

  Range: 6–32

  Interactions:
  When *x* is set to *N*, no separator is specified. The width range is then 5–32, and the default changes to 7.

  When the value of *w* is too small to write a four-digit year, the date appears with only the last two digits of the year.

Details

The YYQR*xw*. format writes SAS date values in one of the following forms:

<yy>yyqr

<yy>yyxqr

where

<yy>yy
: is a two-digit or four-digit integer that represents the year.

x
: is a specified separator.

qr
: is a roman numeral (I, II, III, or IV) that represents the quarter of the year.

Example

The following examples use the input value of 19127, which is the SAS date value that corresponds to May 14, 2012.

SAS Statement	Result
	`----+----1----+`
`put date yyqrc6.;`	`12:II`

SAS Statement	Result
`put date yyqrd.;`	`2012-II`
`put date yyqrn5.;`	`12II`
`put date yyqrp8.;`	`2012.II`
`put date yyqrs10.;`	`  2012/II`

See Also

Formats:

- "YYQxw. Format" on page 188
- "YYQRw. Format" on page 190

Zw.d Format

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

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Default: 0

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 might 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.

Example

```
put @5 seqnum z8.;
```

Value of `seqnum`	Result
	`----+----1`
1350	00001350

ZD*w.d* Format

Writes numeric data in zoned decimal format .

Category: Numeric

Alignment: left

See: "ZDw.d Format: UNIX" in *SAS Companion for UNIX Environments*
"ZDw.d Format: Windows" in *SAS Companion for Windows*
"ZDw.d Format: z/OS" in *SAS Companion for z/OS*

Syntax

ZD*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 1

Range: 1–32

d

specifies to multiply the number by 10^d. This argument is optional.

Default: 0

Range: 0–31

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 type 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

Example

```
y=put(x,zd4.);
put y $hex8.;
```

Value of `x`	Result *
`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.

Part 2

SAS Informats

Chapter 3

About Informats

Definition of Informats

An informat is a type of SAS language element that applies a pattern to or executes instructions for a data value to be read as input. Types of informats correspond to the data's type: numeric, character, date, time, or timestamp. The ability to create user-defined informats is also supported. Examples of SAS informats are BINARY, DATE. and COMMA. For example, the following value contains a dollar sign and commas:

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

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

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

Syntax

SAS informats have the following form:

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

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

informat
: names the informat. The informat is a SAS informat or a user-defined informat that was previously defined with the INVALUE statement in PROC FORMAT. Chapter 23, "FORMAT Procedure" in *Base SAS Procedures Guide*.

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

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

Note: Even though SAS can read up to 32 digits when you specify some numeric informats, numbers with more than 15 significant digits might lose precision due to the limitations of the eight-byte floating-point representation used by most computers.

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

If the informat width is too narrow to read all the columns in the input data, you might get unexpected results. The problem frequently occurs with the date and time informats. You must adjust the width of the informat to include blanks or special characters between the day, month, year, or time. For more information about date and time values, see "Dates, Times, and Intervals" in Chapter 7 of *SAS Language Reference: Concepts*.

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

Using Informats

Ways to Specify Informats

Overview of Specifying Informats

You can specify informats in the following ways:

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

INPUT Statement

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

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

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

For a complete discussion of the INPUT statement, see “INPUT Statement” in *SAS Statements: Reference*.

INPUT Function

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

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

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

Use the PUT function with a SAS format to convert numeric values to character values. For an example of a numeric-to-character conversion, see “PUT Function” in *SAS Functions and CALL Routines: Reference*. For a complete discussion of the INPUT function, see “INPUT Function” in *SAS Functions and CALL Routines: Reference*.

INFORMAT Statement

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

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

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

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

- for numeric informats, the *d* value in an informat behaves in the usual way for numeric informats.

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

See “INPUT Statement, List” in *SAS Statements: Reference* for more information about how to use modified list input to read data.

Note: Any time a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option in either ASCII or EBCDIC environments. For example, when you read an EBCDIC text file on an ASCII platform, it is recommended that you specify the ENCODING= option in the FILENAME or INFILE statement. However, if you use the DSD and the DLM= or DLMSTR= options in the FILENAME or INFILE statement, the ENCODING= option is a requirement because these options require certain characters in the session encoding (such as quotation marks, commas, and blanks).The use of encoding-specific informats should be reserved for use with true binary files. That is, they contain both character and non-character fields.

ATTRIB Statement

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

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

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

For more information, see “ATTRIB Statement” in *SAS Statements: Reference*.

Permanent versus Temporary Association

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

User-Defined Informats

In addition to the informats that are supplied with Base SAS software, you can create your own informats. In Base SAS software, PROC FORMAT enables you to create your own informats and formats for both character and numeric variables. For more information about user-defined informats, see Chapter 23, “FORMAT Procedure” in *Base SAS Procedures Guide*.

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

- to create permanent, not temporary, informats with PROC FORMAT

- to store the source code that creates the informats (the PROC FORMAT step) with the SAS program that uses them.

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

System Options	Result
FMTERR	SAS produces an error that causes the current DATA or PROC step to stop.
NOFMTERR	SAS continues processing by substituting a default informat.

Although using NOFMTERR enables SAS to process a variable, you lose the information that the user-defined informat supplies. This option can cause a DATA step to misread data, and it can produce incorrect results. For more information, see “FMTERR System Option” in *SAS System Options: Reference*.

To avoid problems, make sure that users of your program have access to all the user-defined informats that are used.

Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms

Definitions

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

The ordering of bytes differs between the “big endian” and the “little endian” platforms. These colloquial terms are used to describe byte ordering for IBM mainframes (big endian) and for Intel-based platforms (little endian). In the SAS System, the following platforms are considered big endian: IBM mainframe, HP-UX, AIX, Solaris on SPARC, and Macintosh. In SAS, the following platforms are considered little endian: Intel ABI, Linux, OpenVMS Alpha, OpenVMS Integrity, Solaris on x64, Tru64 UNIX, and Windows.

How the Bytes Are Ordered

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

If an integer is negative, the “two's complement” representation is used. The high-order bit of the most significant byte of the integer will be set on. For example, –2 would be represented in one, two, and four bytes on big endian platforms as FE, FF FE, and FF FF FF FE respectively. On little endian platforms, the representation would be FE, FE FF, and FE FF FF FF. These representations result from the output of the integer binary value –2 expressed in hexadecimal representation.

Reading Data Generated on Big Endian or Little Endian Platforms

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

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

Table 3.1 *SAS Informats and Byte Ordering*

Platform for Which the Data Was Created	Platform the Data Is Read on	Signed Integer	Informat
big endian	big endian	yes	IB or S370FIB
big endian	big endian	no	PIB, S370FPIB, S370FIBU
big endian	little endian	yes	IBR
big endian	little endian	no	PIBR
little endian	big endian	yes	IBR
little endian	big endian	no	PIBR
little endian	little endian	yes	IB or IBR
little endian	little endian	no	PIB or PIBR
big endian	either	yes	S370FIB
big endian	either	no	S370FPIB
little endian	either	yes	IBR
little endian	either	no	PIBR

Integer Binary Notation in Different Programming Languages

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

Language	2 Bytes or 8-Bit Systems	4 Bytes or 16-Bit Systems	8 Bytes or 64-Bit Systems
SAS	IB2., IBR2., PIB2.,PIBR2., S370FIB2., S370FIBU2., S370FPIB2.	IB4., IBR4., PIB4., PIBR4., S370FIB4., S370FIBU4., S370FPIB4.	IB8., IBR8., PIB8., PIBR8., S370FIB8., S370FIBU8., S370FPIB8.
C	short	int	long *
Java	short	int	long *
Visual Basic 6.0	short	long*	none
Visual Basic.NET	short	integer	long *
PL/I	fixed bin(15)	fixed bin(31)	fixed bin(63)
Fortran	integer*2	integer*4	integer*8
COBOL	comp pic 9(4)	comp pic 9(8)	comp pic 9(16)
IBM assembler	H	F	FD

* The size of integers declared as long depends on the operating environment.

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 might 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 character. For example, the value 15 is stored in two nibbles, using the hexadecimal characters 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 character. 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= system option 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 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 and Informats
PIC S9(X) PACKED-DECIMAL	S370FPDw.
PIC 9(X) PACKED-DECIMAL	S370FPDUw.
PIC S9(W) DISPLAY	S370FZDw.
PIC 9(W) DISPLAY	S370FZDUw.
PIC S9(W) DISPLAY SIGN LEADING	S370FZDLw.
PIC S9(W) DISPLAY SIGN LEADING SEPARATE	S370FZDSw.
PIC S9(W) DISPLAY SIGN TRAILING SEPARATE	S370FZDTw.

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

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

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

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

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

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

Summary of Packed Decimal and Zoned Decimal Formats and Informats

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

Format	Data Type 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 (x'40') to EBCDIC zero (x'F0'), and then corresponds to the informat as zoned decimal
none	Zoned decimal	ZDV	Non-IBM zoned decimal representation
S370FPD	Packed decimal	S370FPD	Last nibble C (positive) or D (negative)
S370FPDU	Packed decimal	S370FPDU	Last nibble always F (positive)
S370FZD	Zoned decimal	S370FZD	Last byte contains sign in upper nibble: C (positive) or D (negative)
S370FZDU	Zoned decimal	S370FZDU	Unsigned; sign nibble always F
S370FZDL	Zoned decimal	S370FZDL	Sign nibble in first byte in informat; separate leading sign byte of x'C0' (positive) or x'D0' (negative) in format
S370FZDS	Zoned decimal	S370FZDS	Leading sign of - (x'60') or + (x'4E')
S370FZDT	Zoned decimal	S370FZDT	Trailing sign of - (x'60') or + (x'4E')

Format	Data Type Representation	Corresponding Informat	Comments
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: *mmsstttF*
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

Reading Dates and Times by Using the ISO 860 Basic and Extended Notations

ISO 8601 Formatting Symbols

The following list explains the formatting symbols that are used to notate the ISO 8601 dates, time, datetime, durations, and interval values:

n

specifies a number that represents the number of years, months, or days

P

indicates that the duration that follows is specified by the number of years, months, days, hours, minutes, and seconds

T

indicates that a time value follows. Any value with a time must begin with T.

Requirement: Time values that are read by the extended notation informats that begin with the characters E8601 must use an uppercase T.

W
: indicates that the duration is specified in weeks.

Z
: indicates that the time value is the time in Greenwich, England, or UTC time.

+|-
: the + indicates the time zone offset to the east of Greenwich, England. The - indicates the time zone offset to the west of Greenwich, England.

yyyy
: specifies a four-digit year

mm
: as part of a date, specifies a two-digit month, 01–12

dd
: specifies a two-digit day, 01–1

hh
: specifies a two-digit hour, 00–24

mm
: as part of a time, specifies a two-digit minute, 00–59

ss
: specifies a two-digit second, 00–59

fff* | *ffffff
: specifies an optional fraction of a second using the digits 0–9:

fff	use 1 - 3 digits for values read by the $N8601B informat and the $N8601E informat
ffffff	use 1 - 6 digits for informat other than the $N8601B informat and the $N8601E informat

Y
: indicates that a year value proceeds this character in a duration

M
: as part of a date, indicates that a month value proceeds this character in a duration

D
: indicates that a day value proceeds this character in a duration

H
: indicates that an hour value proceeds this character in a duration

M
: as part of a time, indicates that a minute value proceeds this character in a duration

S
: indicates that a seconds value proceeds this character in a duration

Reading ISO 8601 Date, Time, and Datetime Values

SAS reads ISO 8601 dates, times, and datetimes using various informats, and the resulting values are SAS date, time, or datetime values. The following table shows different date, time, and datetime forms and the informats that you use to read them:

Date, Time, or Datetime	ISO 8601 Notation	Example	Informat
Basic Notations			
Date	YYYYMMDD	20120915	B8601DA*w*.
Time	hhmmssnnnnnn	155300322348	B8601TM*w.d*
Time with time zone	hhmmss+|-hhmm	155300+0500	B8601TZ*w.d*
	hhmmssZ	155300Z	B8601TZ*w.d*
Convert to local time with time zone	hhmmss+|-hhmm	155300+0500	B8601TZ*w.d*
Datetime	YYYYMMDDThhmmssn nnnnn	20120915T155300	B8601DT*w.d*
Datetime with timezone	YYYYMMDDThhmmss +|-hhmm	20120915T155300+0 500	B8601DZ*w.d*
	YYYYMMDDThhmmssZ	20120915T155300Z	B8601DZ*w.d*
Read the date from a datetime	YYYYMMD	20120915	B8601DN*w*.
Extended Notations			
Date	YYYY-MM-DD	2012-09-15	E8601DA*w*.
Time	hh:mm:ss.nnnnnn	15:53:00.322348	E8601TM*w.d*
Time with time zone	hh:mm:ss.nnnnnn+|-hh:mm	15:53:00+05:00	E8601TZ*w.d*
Convert to local time with time zone	hh:mm:ss.nnnnnn+|-hh:mm	15:53:00+05:00	E8601LZ*w.d*
Datetime	YYYY-MM-DDThh:mm:ss.nnnnnn	2012-09-15T15:53:0 0	E8601DT*w.d*
Datetime with time zone	YYYY-MM-DDThh:mm:ss.nnnnnn+|-hh:mm	2012-09-15T15:53:0 0+05:00	E8601DZ*w.d*
Read date from a datetime	YYYY-MM-DD	2012-09-15	E8601DN*w*.

When SAS reads an ISO 8601 value that specifies a time zone offset (+|-*hh*:*mm* or +|-*hhmm*), the time or datetime value is adjusted to account for the offset. A SAS time or datetime value for an ISO 8601 value with a time zone offset is the time or datetime for the zero meridian (Greenwich, England). For example, if SAS reads the datetime 2011–09–15T15:53:00+05:00 using the E8601DZ informat, the datetime value 1631703180 has been adjusted for the five hour time zone difference. This datetime value is the datetime value for the zero meridian. If you write this value using the E8601DZ format, the value is 2011–09–15T10:53:00+00:00. The hour specified after the T shows the five hour adjustment.

Reading ISO 8601 Duration, Interval, and Datetime Values

Informats That Read Duration, Interval, and Datetime Values

SAS uses two informats that reads ISO datetime, duration, and interval values.

$N8601B informat
: reads duration, interval, and datetime values that are specified in either the basic notation or the extended notation

$N8601E informat
: reads duration, interval, and datetime values that are specified only in the extended notation

Use the $N8601E informat when you want to make sure that you are in compliance with the extended notation.

The datetime values that are read by these informats result in a SAS character representation. If you want a datetime value to be read as a numeric value, use the B8601DT informat, the B8601DZ informat, the E8601DT informat, or the E8601DZ informat.

Complete Duration, Interval, and Datetime Notations

The following table shows the formatting of duration, datetime, and interval values that can be read in the complete form:

Table 3.2 Complete Component Forms

Time Component	ISO 8601 Notation	Example
Duration - Basic Notation	PYYYYMMDDThhmmss	P20120915T155300
	-PYYYYMMDDThhmmss	-P20080915T155300
Duration - Extended Notation	PYYYY-MM-DDThh:mm:ss	P2012-09-15T15:53:00
	-PYYYY-MM-DDThh:mm:ss	-P2012-09-15T15:53:00
Duration - Basic and Extended Notation	PnYnMnDTnHnMnS	P2y10m14dT20h13m45s
	-PnYnMnDTnHnMnS	-P2n10m14dT20h13m45s
	PnW (weeks)	P6w
Interval - Basic Notation	YYYYMMDDThhmmss/YYYYMMDDThhmmss	20120915T155300/20141113T000000
	PnYnMnDTnHnMnS/YYYYMMDDThhmmss	P2y10M14dT20h13m45s/20120915T155300

Time Component	ISO 8601 Notation	Example
	YYYYMMDDThhmmss/ PnYnMnDTnHnMnS	20120915T155300/ P2y10M14dT20h13m45s
Interval- Extended Notation	YYYY-MM-DDThh:mm:ss/ YYYY-MM-DDThh:mm:ss	2012-09-15T15:53:00/2014-1 1-13T00:00:00
	PnYnMnDTnHnMnS/ YYYY-MM-DDThh:mm:ss	P2y10M14dT20h13m45s/ 2012-09-15T15:53:00
	YYYY-MM-DDThh:mm:ss/ PnYnMnDTnHnMnS	2012-09-15T15:53:00/ P2y10M14dT20h13m45s
Datetime-Basic Notation	YYYYMMDDThhmmss.fff +|-hhmm	20120915T155300
	(all blank)	
Datetime-Extended Notation	YYYY-MM- DDThh:mm:ss.fff+|-hhmm	2012-09-15T15:53:00 +04:30
	(all blank)	

Reading Omitted Components

One or more date or time components can be omitted from a datetime value or a duration value that is in the form P*yyyymmdd*. SAS reads omitted components using the $N8601B informat or the $N8601E informat, and the omitted component must be represented by a hyphen (-).

The following examples show duration, datetime, and interval values with omitted components:

`p0003-02--T10:31:33`
The omitted component is the number of days.

`-p0003-02-02T-:31:33`
The omitted component is the number of hours.

`x-09-15T15:x:x`
The omitted components are the number of years, minutes, and seconds.

`2012-09-15T15:x:00/2010-09-15T15:x:00`
The omitted components are the minutes.

When reading values that contain a time zone offset, omitted components are not allowed. Use 00 in place of omitted components.

Truncated Values

SAS reads truncated duration, datetime, and interval values, where one or more lower order components is truncated because the value is 0 or the value is not significant.

The following list shows examples of truncated values:

- **`p00030202T1031`**
- **`2012-09-15T15/2014-09-15T15:53`**

- **-p0003-03-03T-:-:-**
- **P2y3m4dT5h6m**
- **2012-09-xTx:x:x**
- **2012**

When reading values that contain a time zone offset, truncation is not allowed. Use 00 in place of truncated values.

Normalizing Duration Components

When a value for a duration component is greater than the largest standard value for a component, SAS normalizes the component except when the duration component is a single component. The following table shows examples of normalized duration components:

Duration	Extended Normalized Duration
p3y13m	p0004-01
pt24h24m65s	P----01T-:25:05
p3y13mT24h61m	P0004-01-01T01:01
p0004-13	p0005-01
p0003-02-61T15:61:61	P0003-04-01T16:02:01
p13m	P13M

If a component contains the largest value, such as 60 for minutes or seconds, SAS normalizes the value and replaces the value with a hyphen. For example, **pT12:60:13** becomes **PT13:-:13**.

Thirty days is used to normalize a month.

Dates and times in a datetime value that are greater than the standard value for the component are not normalized. They produce an error.

Fractions in Durations, Datetime, and Interval Values

Ending components can contain a fraction that consists of a period or a comma, followed by one to three digits. The following examples show the use of fractions in duration, datetime, and interval values:

- **201209.5**
- **P2012-09-15T10.33**
- **2012-09-15/P0003-03-03,333**

Chapter 4

Dictionary of Informats

Informats Documented in Other Publications

For informats that support national language, see Chapter 12, “Informat Entries,” in *SAS National Language Support (NLS): Reference Guide*.

Informats by Category

There are five categories of informats in this list:

Category	Description
Character	instructs SAS to read character data values into character variables.
Column Binary	instructs SAS to read data stored in column-binary or multipunched form into character or numeric variables.
Date and Time	instructs SAS to read date values into variables that represent dates, times, and datetimes.
ISO 8601	instructs SAS to read date, time, and datetime values that are written in the ISO 8601 standard into either numeric or character variables.
Numeric	instructs SAS to read numeric data values into numeric variables.

For information about column-binary data, see “Reading Column-Binary Data” in Chapter 19 of *SAS Language Reference: Concepts*. For information about creating user-defined informats, see Chapter 23, “FORMAT Procedure” in *Base SAS Procedures Guide*.

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

Category	Language Elements	Description
Character	$ASCIIw. Informat (p. 224)	Converts ASCII character data to native format.
	$BASE64Xw. Informat (p. 225)	Converts ASCII text into character data by using Base 64 encoding.

Category	Language Elements	Description
	$BINARYw. Informat (p. 226)	Converts binary data to character data.
	$CHARw. Informat (p. 228)	Reads character data with blanks.
	$CHARZBw. Informat (p. 229)	Converts binary 0s to blanks.
	$EBCDICw. Informat (p. 230)	Converts EBCDIC character data to native format.
	$HEXw. Informat (p. 231)	Converts hexadecimal data to character data.
	$OCTALw. Informat (p. 236)	Converts octal data to character data.
	$PHEXw. Informat (p. 237)	Converts packed hexadecimal data to character data.
	$QUOTEw. Informat (p. 238)	Removes matching quotation marks from character data.
	$UPCASEw. Informat (p. 239)	Converts character data to uppercase.
	$VARYINGw. Informat (p. 239)	Reads character data of varying length.
	$w. Informat (p. 241)	Reads standard character data.
Column Binary	$CBw. Informat (p. 227)	Reads standard character data from column-binary files.
	CBw.d Informat (p. 264)	Reads standard numeric values from column-binary files.
	PUNCH.d Informat (p. 308)	Reads whether a row of column-binary data is punched.
	ROWw.d Informat (p. 313)	Reads a column-binary field down a card column.
Date and Time	$N8601Bw.d Informat (p. 232)	Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notations.
	$N8601Ew.d Informat (p. 234)	Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.
	ANYDTDTEw. Informat (p. 242)	Reads and extracts the date value from various date, time, and datetime forms.
	ANYDTDTMw. Informat (p. 245)	Reads and extracts datetime values from various date, time, and datetime forms.
	ANYDTTMEw. Informat (p. 248)	Reads and extracts time values from various date, time, and datetime forms.
	B8601CIw.d Informat (p. 250)	Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss<fff>.
	B8601DAw. Informat (p. 252)	Reads date values that are specified using the ISO 8601 base notation yyyymmdd.

<table>
<tr><th>Category</th><th>Language Elements</th><th>Description</th></tr>
<tr><td></td><td>B8601DJw.d Informat (p. 253)</td><td>Reads a Java date and time value that is in the form yyyymmddhhmmss<ffffff>.</td></tr>
<tr><td></td><td>B8601DNw. Informat (p. 254)</td><td>Reads date values that are specified using the ISO 8601 basic notation yyyymmdd and returns SAS datetime values where the time portion of the value is 000000.</td></tr>
<tr><td></td><td>B8601DTw.d Informat (p. 255)</td><td>Reads datetime values that are specified using the ISO 8601 basic notation yyyymmddThhmmss<ffffff>.</td></tr>
<tr><td></td><td>B8601DZw.d Informat (p. 257)</td><td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation yyyymmddThhmmss+|–hhmm or yyyymmddThhmmss<ffffff>Z.</td></tr>
<tr><td></td><td>B8601TMw.d Informat (p. 258)</td><td>Reads time values that are specified using the ISO 8601 basic notation hhmmss<ffffff>.</td></tr>
<tr><td></td><td>B8601TZw.d Informat (p. 259)</td><td>Reads time values that are specified using the ISO 8601 basic time notation hhmmss<ffff>+|–hhmm or hhmmss<ffffff>Z.</td></tr>
<tr><td></td><td>DATEw. Informat (p. 267)</td><td>Reads date values in the form ddmmmyy or ddmmmyyyy.</td></tr>
<tr><td></td><td>DATETIMEw. Informat (p. 268)</td><td>Reads datetime values in the form ddmmmyy hh:mm:ss.ss or ddmmmyyyy hh:mm:ss.ss.</td></tr>
<tr><td></td><td>DDMMYYw. Informat (p. 270)</td><td>Reads date values in the form ddmmyy<yy> or dd-mm-yy<yy>, where a special character, such as a hyphen (-), period (.), or slash (/), separates the day, month, and year; the year can be either 2 or 4 digits.</td></tr>
<tr><td></td><td>E8601DAw. Informat (p. 272)</td><td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.</td></tr>
<tr><td></td><td>E8601DNw. Informat (p. 273)</td><td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd and returns SAS datetime values where the time portion of the value is 000000.</td></tr>
<tr><td></td><td>E8601DTw.d Informat (p. 274)</td><td>Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.<ffffff>.</td></tr>
<tr><td></td><td>E8601DZw.d Informat (p. 276)</td><td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime extended notation yyyy-mm-ddThh:mm:ss+|–hh:mm.<fffff> or yyyy-mm-ddThh:mm:ss.<fffff>Z.</td></tr>
<tr><td></td><td>E8601LZw.d Informat (p. 277)</td><td>Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation hh:mm:ss+|–hh:mm.<fffff> or hh:mm:ss.<fffff>Z and converts the values to the local time.</td></tr>
<tr><td></td><td>E8601TMw.d Informat (p. 279)</td><td>Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.<ffffff>.</td></tr>
</table>

Category	Language Elements	Description
	E8601TZw.d Informat (p. 280)	Reads time values that are specified using the ISO 8601 extended time notation hh:mm:ss+|–hh:mm.<ffffff> or hh:mm:ssZ.
	HHMMSSw. Informat (p. 284)	Reads hours, minutes, and seconds in the form hh:mm:ss or hhmmss.
	JULIANw. Informat (p. 289)	Reads Julian dates in the form yyddd or yyyyddd.
	MDYAMPMw.d Informat (p. 291)	Reads datetime values in the form mm-dd-yy<yy> hh:mm:ss.ss AM|PM, where a special character such as a hyphen (-), period (.), slash (/), or colon (:) separates the month, day, and year; the year can be either 2 or 4 digits.
	MMDDYYw. Informat (p. 292)	Reads date values in the form mmddyy or mmddyyyy.
	MONYYw. Informat (p. 294)	Reads month and year date values in the form mmmyy or mmmyyyy.
	MSECw. Informat (p. 295)	Reads TIME MIC values.
	PDJULGw. Informat (p. 300)	Reads packed Julian date values in the hexadecimal form yyyydddF for IBM.
	PDJULIw. Informat (p. 301)	Reads packed Julian dates in the hexadecimal format ccyydddF for IBM.
	PDTIMEw. Informat (p. 302)	Reads packed decimal time of SMF and RMF records.
	RMFDURw. Informat (p. 311)	Reads duration intervals of RMF records.
	RMFSTAMPw. Informat (p. 312)	Reads time and date fields of RMF records.
	SHRSTAMPw. Informat (p. 329)	Reads date and time values of SHR records.
	SMFSTAMPw. Informat (p. 330)	Reads time and date values of SMF records.
	STIMERw. Informat (p. 331)	Reads time values and determines whether the values are hours, minutes, or seconds; reads the output of the STIMER system option.
	TIMEw. Informat (p. 332)	Reads hours, minutes, and seconds in the form hh:mm:ss.ss, where special characters such as the colon (:) or the period (.) are used to separate the hours, minutes, and seconds.
	TODSTAMPw. Informat (p. 334)	Reads an eight-byte time-of-day stamp.
	TUw. Informat (p. 335)	Reads timer units.

Category	Language Elements	Description
	WEEKUw. Informat (p. 340)	Reads a value in the form of a week-number within the year and returns a SAS date value by using the U algorithm.
	WEEKVw. Informat (p. 341)	Reads a value in the form a week-number within a year and returns a SAS date value using the V algorithm.
	WEEKWw. Informat (p. 343)	Reads a value in the form of a week-number within the year and returns a SAS date value using the W algorithm.
	YMDDTTMw.d Informat (p. 345)	Reads datetime values in the form <yy>yy-mm-dd hh:mm:ss.ss, where special characters such as a hyphen (-), period (.), slash (/), or colon (:) are used to separate the year, month, day, hour, minute, and seconds; the year can be either 2 or 4 digits.
	YYMMDDw. Informat (p. 347)	Reads date values in the form yymmdd or yyyymmdd.
	YYMMNw. Informat (p. 348)	Reads date values in the form yyyymm or yymm.
	YYQw. Informat (p. 350)	Reads quarters of the year in the form yyQq or yyyyQq.
ISO 8601	$N8601Bw.d Informat (p. 232)	Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notations.
	$N8601Ew.d Informat (p. 234)	Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.
	B8601CIw.d Informat (p. 250)	Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss<fff>.
	B8601DAw. Informat (p. 252)	Reads date values that are specified using the ISO 8601 base notation yyyymmdd.
	B8601DJw.d Informat (p. 253)	Reads a Java date and time value that is in the form yyyymmddhhmmss<ffffff>.
	B8601DNw. Informat (p. 254)	Reads date values that are specified using the ISO 8601 basic notation yyyymmdd and returns SAS datetime values where the time portion of the value is 000000.
	B8601DTw.d Informat (p. 255)	Reads datetime values that are specified using the ISO 8601 basic notation yyyymmddThhmmss<ffffff>.
	B8601DZw.d Informat (p. 257)	Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation yyyymmddThhmmss+|–hhmm or yyyymmddThhmmss<ffffff>Z.
	B8601TMw.d Informat (p. 258)	Reads time values that are specified using the ISO 8601 basic notation hhmmss<ffffff>.
	B8601TZw.d Informat (p. 259)	Reads time values that are specified using the ISO 8601 basic time notation hhmmss<ffff>+|–hhmm or hhmmss<ffffff>Z.

Category	Language Elements	Description
	E8601DAw. Informat (p. 272)	Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.
	E8601DNw. Informat (p. 273)	Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd and returns SAS datetime values where the time portion of the value is 000000.
	E8601DTw.d Informat (p. 274)	Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.<ffffff>.
	E8601DZw.d Informat (p. 276)	Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime extended notation yyyy-mm-ddThh:mm:ss+|–hh:mm.<fffff> or yyyy-mm-ddThh:mm:ss.<fffff>Z.
	E8601LZw.d Informat (p. 277)	Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation hh:mm:ss+|–hh:mm.<fffff> or hh:mm:ss.<fffff>Z and converts the values to the local time.
	E8601TMw.d Informat (p. 279)	Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.<ffffff>.
	E8601TZw.d Informat (p. 280)	Reads time values that are specified using the ISO 8601 extended time notation hh:mm:ss+|–hh:mm.<ffffff> or hh:mm:ssZ.
Numeric	BINARYw.d Informat (p. 261)	Converts positive binary values to integers.
	BITSw.d Informat (p. 262)	Extracts bits.
	BZw.d Informat (p. 263)	Converts blanks to 0s.
	COMMAw.d Informat (p. 265)	Removes embedded characters.
	COMMAXw.d Informat (p. 266)	Removes embedded periods, blanks, dollar signs, percent signs, dashes, and closing parenthesis from the input data. An open parenthesis at the beginning of a field is converted to a minus sign. The COMMAX informat reverses the roles of the decimal point and the comma.
	Ew.d Informat (p. 272)	Reads numeric values that are stored in scientific notation and double-precision scientific notation.
	FLOATw.d Informat (p. 282)	Reads a native single-precision, floating-point value and divides it by 10 raised to the dth power.
	HEXw. Informat (p. 283)	Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values.
	IBw.d Informat (p. 286)	Reads native integer binary (fixed-point) values, including negative values.
	IBRw.d Informat (p. 287)	Reads integer binary (fixed-point) values in Intel and DEC formats.

Category	Language Elements	Description
	IEEEw.d Informat (p. 288)	Reads an IEEE floating-point value and divides it by 10 raised to the d th power.
	NUMXw.d Informat (p. 296)	Reads numeric values with a comma in place of the decimal point.
	OCTALw.d Informat (p. 297)	Converts positive octal values to integers.
	PDw.d Informat (p. 298)	Reads data that are stored in IBM packed decimal format.
	PERCENTw.d Informat (p. 303)	Reads percentages as numeric values.
	PIBw.d Informat (p. 304)	Reads positive integer binary (fixed-point) values.
	PIBRw.d Informat (p. 306)	Reads positive integer binary (fixed-point) values in Intel and DEC formats.
	PKw.d Informat (p. 307)	Reads unsigned packed decimal data.
	RBw.d Informat (p. 309)	Reads numeric data that are stored in real binary (floating-point) notation.
	S370FFw.d Informat (p. 315)	Reads EBCDIC numeric data.
	S370FIBw.d Informat (p. 316)	Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format.
	S370FIBUw.d Informat (p. 317)	Reads unsigned integer binary (fixed-point) values in IBM mainframe format.
	S370FPDw.d Informat (p. 318)	Reads packed data in IBM mainframe format.
	S370FPDUw.d Informat (p. 319)	Reads unsigned packed decimal data in IBM mainframe format.
	S370FPIBw.d Informat (p. 320)	Reads positive integer binary (fixed-point) values in IBM mainframe format.
	S370FRBw.d Informat (p. 322)	Reads real binary (floating-point) data in IBM mainframe format.
	S370FZDw.d Informat (p. 323)	Reads zoned decimal data in IBM mainframe format.
	S370FZDBw.d Informat (p. 324)	Reads zoned decimal data in which zeros have been left blank.
	S370FZDLw.d Informat (p. 325)	Reads zoned decimal leading-sign data in IBM mainframe format.
	S370FZDSw.d Informat (p. 326)	Reads zoned decimal separate leading-sign data in IBM mainframe format.
	S370FZDTw.d Informat (p. 327)	Reads zoned decimal separate trailing-sign data in IBM mainframe format.

Category	Language Elements	Description
	S370FZDUw.d Informat (p. 328)	Reads unsigned zoned decimal data in IBM mainframe format.
	TRAILSGNw. Informat (p. 334)	Reads a trailing plus (+) or minus (–) sign.
	VAXRBw.d Informat (p. 336)	Reads real binary (floating-point) data in VMS format.
	VMSZNw.d Informat (p. 337)	Reads VMS and MicroFocus COBOL zoned numeric data.
	w.d Informat (p. 338)	Reads standard numeric data.
	ZDw.d Informat (p. 351)	Reads zoned decimal data.
	ZDBw.d Informat (p. 353)	Reads zoned decimal data in which zeros have been left blank.
	ZDVw.d Informat (p. 353)	Reads and validates zoned decimal data.

Dictionary

$ASCII*w*. Informat

Converts ASCII character data to native format.

Category: Character

Syntax

$ASCII*w*.

Syntax Description

w

specifies the width of the input field.

Default: 1 if the length of the variable is undefined. Otherwise, the default is the length of the variable.

Range: 1–32767

Details

If ASCII is the native format, no conversion occurs.

Comparisons

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

Example

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

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

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

$BASE64X*w*. Informat

Converts ASCII text into character data by using Base 64 encoding.

Category: Character

Alignment: left

Syntax

$BASE64X*w*.

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–32767

Details

Base 64 is an industry encoding method whose encoded characters are determined by using a positional scheme that uses only ASCII characters. Several Base 64 encoding schemes have been defined by the industry for specific uses, such as e-mail or content masking. SAS maps positions 0–61 to the characters A–Z, a–z, and 0–9. Position 62 maps to the character +, and position 63 maps to the character /.

The following are some uses of Base 64 encoding:

- embed binary data in an XML file
- encode passwords
- encode URLs

The '=' character in the encoded results indicates that the results have been padded with zero bits. In order for the encoded characters to be decoded, the '=' must be included in the value to be decoded.

Example

```
input @1 b64exmpl $base64x64.;
```

Data Line	Result
RkNBMDFBNzk5M0JD	FCA01A7993BC
TXlQYXNzd29yZA==	MyPassword
d3d3Lm15ZG9tYWluLmNvbi9teWhpZGRlblVSTA==	www.mydomain.com/ myhiddenURL

See Also

- The XMLDOUBLE option of the "LIBNAME Statement Syntax" in *SAS XML LIBNAME Engine: User's Guide*

Formats:

- "$BASE64Xw. Format" on page 34

$BINARY*w*. Informat

Converts binary data to character data.

Category: Character

Syntax

$BINARY*w*.

Syntax Description

w

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

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

Default: 8

Range: 1–32767

Details

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

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

Comparisons

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

Example

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

Data Line	Result	
----+----1----+----2	ASCII	EBCDIC
0100110001001101	LM	<(

$CB*w*. Informat

Reads standard character data from column-binary files.

Category: Column Binary

Syntax

$CB*w*.

Syntax Description

w

specifies the width of the input field.

Default: none

Range: 1–32767

Details

Column-binary data storage compresses data so that more than 80 items of data can be stored on a single “virtual” punch card.

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

Example

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

Data Line *	Result	
`----+----1`	EBCDIC	ASCII
`200A`	+	N

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

See Also

- “How to Read Column-Binary Data” in Chapter 19 of *SAS Language Reference: Concepts*

Informats:

- “CBw.d Informat” on page 264
- “PUNCH.d Informat” on page 308
- “ROWw.d Informat” on page 313

$CHAR*w*. Informat

Reads character data with blanks.

Category: Character

Syntax

$CHAR*w*.

Syntax Description

w

specifies the width of the input field.

Default: 8 if the length of the variable is undefined. Otherwise, the default is the length of the variable

Range: 1–32767

Details

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

Comparisons

- The $CHAR*w*. informat is almost identical to the $*w*. informat. However $CHAR*w*. does not trim leading blanks or convert a single period in the input data field to a blank, while the $*w*. informat does.

- Use the table below to compare the SAS informat $CHAR8. with notation in other programming languages:

Language	Character Notation
SAS	$CHAR8.
IBM 370 assembler	CL8
C	char [8]
COBOL	PIC x(8)
Fortran	A8
PL/I	CHAR(8)

Example

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

Data Line	Result *
----+----1	
XYZ	XYZ##
XYZ	#XYZ#
.	##.##
X YZ	#X#YZ

* The character # represents a blank space.

$CHARZB*w*. Informat

Converts binary 0s to blanks.

Category: Character

Syntax

$CHARZB*w*.

Syntax Description

w

specifies the width of the input field.

Default: 1 if the length of the variable is undefined. Otherwise, the default is the length of the variable.

Range: 1–32767

Details

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

Comparisons

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

Example

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

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

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

** The character # represents a blank space.

$EBCDIC*w*. Informat

Converts EBCDIC character data to native format.

Category: Character

Syntax

$EBCDIC*w*.

Syntax Description

w

specifies the width of the input field.

Default: 1 if the length of the variable is undefined. Otherwise, the default is the length of the variable.

Range: 1–32767

Details

If EBCDIC is the native format, no conversion occurs.

Note: Any time a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option on either ASCII or EBCDIC environments. When that you read an EBCDIC text file on an ASCII platform, it is recommended that you specify the ENCODING= option in the FILENAME or INFILE statement. However, if you use the DSD and the DLM= or DLMSTR= options in the FILENAME or INFILE statement, the ENCODING= option is a requirement because these options require certain characters in the session encoding (for example, quotation marks, commas, and blanks). The use of encoding-specific informats should be reserved for use with true binary files. That is, they contain both character and non-character fields.

Comparisons

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

Example

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

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

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

$HEX*w*. Informat

Converts hexadecimal data to character data.

Category: Character

See: "$HEXw. Informat: UNIX" in *SAS Companion for UNIX Environments*
"$HEXw. Informat: Windows" in *SAS Companion for Windows*

Syntax

$HEX*w*.

Syntax Description

w

specifies the number of digits of hexadecimal data.

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

Default: 2

Range: 1–32767

Details

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

Comparisons

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

Example

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

Data Line	Result	
`----+----1`	ASCII	EBCDIC
`6C6C`	`ll`	`%%`

$N8601B*w.d* Informat

Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notations.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.4, complete representation

Syntax

$N8601B*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601B informat reads ISO 8601 duration, interval, and datetime values as character data for the following basic notations:

Time Component	ISO 8601 Notation	Example
Duration	P*yyyy-mm-dd*T*hh:mm:ss.fff*	P2012-09-15T15:53:00
	P*yyyymmdd*T*hhmmss*	P00020304T050607
	P*n*Y*n*M*n*DT*n*H*n*M*n.fff*S	P2y10m14dT20h13m45.222s
	P*n*W	P6w
Interval	*yyyy-mm-dd*T*hh:mm:ss.fff*/ *yyyy-mm-dd*T*hh:mm:ss.fff*	2012-09-15T15:53:00/2014-1 1-13T00:00:00
	*yyyymmdd*T*hhmmss.fff*/ *yyyymmdd*T*hhmmss.fff*	20120915T155300/20141115 T120000
	P*n*Y*n*M*n*DT*n*H*n*M*n.fff*S/ *yyyy-mm-dd*T*hh:mm:ss.fff*	P2y10M14dT20h13m45s/ 2012-09-15T15:53:00
	*yyyy-mm-dd*T*hh:mm:ss.fff*/ P*n*Y*n*M*n*DT*n*H*n*M*n.fff*S	2012-09-15T15:53:00/ P2y10M14dT20h13m45s
Datetime	*yyyy-mm-dd*T*hh:mm:ss.fff*	2012-09-15T15:53:00
	yyyymmddThhmmss.fff	20120915T155300

The $N8601B informat also reads ISO 8601 duration, interval, and datetime components that contain omitted or truncated components. Omitted components must use a single hyphen (-) to represent the component.

Comparisons

The $N8601B informat reads durations, intervals, and datetimes that are specified in either the basic or extended notation.

The $N8601E informat reads durations, intervals, and datetimes that are specified only in the extended notation. Use the $N8601E informat when you need to ensure compliance with the extended notation.

Example

```
input @1 i860 $n8601b.;
```

Data Line	Result
p0002-04-05t5:1:12	0002405050112FFC
2012-09-15T15:53:00/2010-09-15T00:00:00	2012915155300FFD2010915000000FFD
p0033-01-04T3:2:55/2012-09-15T15:53:00	0033104030255FFC2012915155300FFD

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

$N8601E*w.d* Informat

Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.

Categories: Date and Time
ISO 8601

Alignment: left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.4, complete representation

Syntax

$N8601E*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 50

Range: 1–200

Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–3

Details

The $N8601E informat reads ISO 8601 duration, interval, and datetime values that can be specified in the following extended notations:

Time Component	ISO 8601 Notation	Example
Duration	P*yyyy-mm-dd*T*hh:mm:ss.fff*	P2012-09-15T15:53:00
	PnW	P6w
Interval	*yyyy-mm-dd*T*hh:mm:ss.fff*/ *yyyy-mm-dd*T*hh:mm:ss.fff*	2012-09-15T15:53:00/2014-11-13T00:00:00
	*yyyy-mm-dd*T*hh:mm:ss.fff*/ P*n*Y*n*M*n*DT*n*H*n*M*ns.fff*S	2012-09-15T15:53:00/ P2Y10M14DT20H13M45S
Datetime	*yyyy-mm-dd*T*hh:mm:ss.fff*	2012-09-15T15:53:00

n
: specifies a number that represents the number of years, months, or days.

P
: is the character that is used to indicate that the duration that follows is specified by the number of years, months, days, hours, minutes, and seconds.

W
: is the character that is used to designate that the duration is specified in weeks.

T
: is the character that is used to designate that a time value follows. If all time values are 0, T is not required.

/
: in an interval, is used to separate the beginning and ending datetime values.

yyyy
: specifies a four-digit year.

mm
: specifies a two-digit month between 01 and 12.

dd
: specifies a two-digit day between 01 and 31.

hh
: specifies a two-digit hour between 00 and 23.

mm
: specifies a two-digit minute between 00 and 59.

ss
: specifies a two-digit second between 00 and 59.

fff
: specifies an optional fraction of a second with a precision of up to three digits, where each digit is between 0 and 9.

Y
: is the character that is used to designate years in a duration.

M
: is the character that is used to designate months in a duration.

D
: is the character that is used to designate days in a duration.

H
: is the character that is used to designate hours in a duration.

M
: is the character that is used to designate minutes in a duration.

S
: is the character that is used to designate seconds in a duration.

Comparisons

The $N8601E informat reads valid durations, intervals, and datetimes that are specified only in the extended notation.

The $N8601B informat reads valid durations, intervals, and datetimes that are specified in either the basic or extended notation.

Use the $N8601E informat when you need to ensure compliance with the extended notation.

Example

```
input @1 i860 $n8601e.;
```

Data Line	Result
p0002-04-05t5:1:12s	0002405050112FFC
2012-09-15T15:53:00/2014-09-15T00:00:00	2012915155300FFD2014915000000FFD
p0033-01-04T3:2:55/2012-09-15T15:53:00	0033104030255FFC2012915155300FFD

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

$OCTAL*w*. Informat

Converts octal data to character data.

Category: Character

Syntax

$OCTAL*w*.

Syntax Description

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

  Default: 3

Range: 1–32767

Details

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

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

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

Comparisons

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

Example

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

Data Line	Result	
----+----1	EBCDIC	ASCII
114	<	L

$PHEX*w*. Informat

Converts packed hexadecimal data to character data.

Category: Character

Syntax

$PHEX*w*.

Syntax Description

w

specifies the number of bytes in the input.

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

Default: 2

Range: 1–32767

Details

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

Comparisons

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

Example

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

Data Line *	Result
0001111000001111	1E0

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

$QUOTE*w*. Informat

Removes matching quotation marks from character data.

Category: Character

Syntax

$QUOTE*w*.

Syntax Description

w

specifies the width of the input field.

Default: 8 if the length of the variable is undefined. Otherwise, the default is the length of the variable.

Range: 1–32767

Example

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

Data Line	Result
----+----1	
'SAS'	SAS

Data Line	Result
"SAS"	SAS
"SAS's"	SAS's

$UPCASE*w*. Informat

Converts character data to uppercase.

Category: Character

Syntax

$UPCASE*w*.

Syntax Description

w

specifies the width of the input field.

Default: 8 if the length of the variable is undefined. Otherwise, the default is the length of the variable.

Range: 1–32767

Details

Special characters, such as hyphens, are not altered.

Example

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

Data Line	Result
----+----1	
sas	SAS

$VARYING*w*. Informat

Reads character data of varying length.

Category: Character

Syntax

$VARYING*w*. *length-variable*

Syntax Description

w

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

Default: 8 if the length of the variable is undefined. Otherwise, the default is the length of the variable.

Range: 1–32767

length-variable

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

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

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

Tips:

If the value of *length-variable* is negative or missing, SAS reads no data from the corresponding record.

If the value of *length-variable* is 0, the value of the variable is a blank character. A value of 0 for *length-variable* enables you to read zero-length records and fields.

If a variable has been read using an informat other than the $VARYING. informat, and then the same data is read into the same variable that uses the $VARYING. informat where *length-variable* is 0, then the previous value is overwritten with a blank value.

If *length-variable* is greater than 0 but less than *w*, SAS reads the number of columns that are specified by *length-variable*. Then SAS pads the value with trailing blanks up to the maximum width that is assigned to the variable.

If *length-variable* is greater than or equal to *w*, SAS reads *w* columns.

Details

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

Examples

Example 1: Obtaining a Current Record Length Directly

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

Data Line	Result *
`----+----1`	
`5shark`	`shark`
`3sunfish`	`sun`

Data Line	Result *
8bluefish	bluefish

* Notice the result of reading the second data line.

Example 2: Obtaining a Record Length Indirectly

Use the LENGTH= option in the INFILE statement to obtain a record length indirectly. The input data lines and results follow the explanation of the SAS statements.

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

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

Data Line	Result
`----+----1----+----2`	
`PATEL    CHEMISTRY`	`PATEL    CHEMISTRY`
`JOHNSON  GEOLOGY`	`JOHNSON  GEOLOGY`
`WILCOX   ART`	`WILCOX   ART`

$*w*. Informat

Reads standard character data.

Category: Character

Alias: $F*w*.

Syntax

$*w*.

Syntax Description

w

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

Range: 1–32767

Details

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

Comparisons

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

Example

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

Data Line	Result *
`----+----1`	
`XYZ`	`XYZ##`
` XYZ`	`XYZ##`
`   .`	
` X YZ`	`X#YZ#`

* The character # represents a blank space.

ANYDTDTE*w*. Informat

Reads and extracts the date value from various date, time, and datetime forms.

Category: Date and Time

Syntax

ANYDTDTE*w*.

Syntax Description

w

specifies the width of the input field.

Default: 9

Range: 5–32

Details

The ANYDTDTE informat reads input data that corresponds to any of the following informats or date, time, or datetime forms and extracts the date part from the derived value.

Informat or Form of Input	Example Data	Informat or Form of Input	Example Data
DATE	01JAN12 01JAN2012	MONYY	JAN12 JAN2012
DATETIME	01JAN12 14:30:08 01JAN2012 14:30:08.5	TIME	14:30 14:30:08.05
DDMMYY	010112 01012012	YMDDTTM	12-01-01 11:23
JULIAN	12001 2012001	YYMMDD	120101 20120101
MDYAMPM	01-01-12 3:53 pm	YYQ	12Q1 2012Q1
MMDDYY	010112 01012012	YY<YY>*x*MM *	12/01 2012-01
MM*x*YY<YY> *	01/12 01-2012	*month-day-year*	January 1, 2012

* *x* is a special character that separates the month from the year.

If the input value is a time-only value, then SAS assumes a date of 01JAN1960.

It is possible for input data such as 01-02-03 or 01-02 to be ambiguous with respect to the month, day, and year. In this case, the DATESTYLE system option indicates the order of the month, day, and year.

Comparisons

The ANYDTDTE informat extracts the date part from the derived value. The ANYDTDTM informat extracts the datetime part. The ANYDTTME informat extracts the time part.

Example

```
input dateinfo anydtdte21.;
```

Data Line	Informat Form	Result	Formatted with the DATE*w.* Format
`----+----1----+----2`			
`01JAN12`	DATE	`18993`	`01JAN12`
`01JAN2012 14:30:08.5`	DATETIME	`18993`	`01JAN12`
`01012012`	DDMMYY	`18993`	`01JAN12`
`2012001`	JULIAN	`18993`	`01JAN12`
`01/01/12`	MMDDYY	`18993`	`01JAN12`
`JAN2012`	MONYY	`18993`	`01JAN12`
`14:30`	TIME	`0`	`01JAN60`
`20120101`	YYMMDD	`18993`	`01JAN12`
`12q1`	YYQ	`18993`	`01JAN12`
`January 1, 2012`	none	`18993`	`01JAN12`

See Also

Informats:

- "ANYDTDTMw. Informat" on page 245
- "ANYDTTMEw. Informat" on page 248
- "DATEw. Informat" on page 267
- "DATETIMEw. Informat" on page 268
- "DDMMYYw. Informat" on page 270
- "JULIANw. Informat" on page 289
- "MDYAMPMw.d Informat" on page 291
- "MMDDYYw. Informat" on page 292
- "MONYYw. Informat" on page 294
- "TIMEw. Informat" on page 332
- "YMDDTTMw.d Informat" on page 345
- "YYMMDDw. Informat" on page 347
- "YYQw. Informat" on page 350

ANYDTDTM*w*. Informat

Reads and extracts datetime values from various date, time, and datetime forms.

Category: Date and Time

Syntax

ANYDTDTM*w*.

Syntax Description

w

specifies the width of the input field.

Default: 19

Range: 1–32

Details

The ANYDTDTM informat reads data that is in the form of any of the following informats or date/time forms, and extracts the datetime part from the derived value:

Informat or Form of Input	Example Data
DATE	01JAN12 01JAN2012
DATETIME	01JAN12 14:30:08 01JAN2012 14:30:08.5
DDMM<YY>YY	010112 01012012
JULIAN	12001 2012001
MMDD<YY>YY *	010112 01012012
MM*x*<YY>YY *	01/12 01-2012
MDYAMPM **	01/01/12 02:30:08 AM 01/01/2012 02:30:08 AM
MONYY	JAN12 JAN2012

Informat or Form of Input	Example Data
TIME	14.30 14:30:08.05
<YY>YYMMDD	120101 20120101
<YY>YYQ	12Q1 2012Q1
<YY>YY*x*MM [*]	12/01 2012/01
month-day-year	January 1, 2012

* *x* is a special character that separates the month from the year. <YY> indicates the century is optional.

** IF AM | PM is not present and the month and day values are ambiguous, the value for the DATESTYLE= system option is used to determine the order.

If the input value is a time-only value, then SAS assumes a date of 01JAN1960. If the input value is a date-only value , then SAS assumes a time of 12:00 midnight. Input time values must include hours and minutes. If any part of a date in the input value is missing in the input value, or if the hour and minutes in a time value are missing or out of range, then the value read is a SAS missing value.

The input values for the preceding informats are mutually exclusive except for MMDDYY, DDMMYY, or YYMMDD when two-digit years are used. It is possible for input data such as 01-02-03 or 01-02 to be ambiguous with respect to the month, day, and year. In this case, the DATESTYLE system option indicates the order of the month, day, and year.

The ANYDTTME informat uses the following rules when reading colons and periods in time values:

Use of Colons and Periods	Example
a single colon in the value *h*:*m* indicates hours and minutes	14:30
two colons in the value h:m:s indicate hours, minutes, and seconds	14:30:08
a single period in the value m:s.ff, where ff is a fraction of a second, indicates that the number preceding the period is the number of seconds	2:39.66
multiple periods in the value indicate that the period is a delimiter for dates and the value is not a time value.	12.25.2012

Comparisons

The ANYDTDTE informat extracts the date part from the derived value. The ANYDTDTM informat extracts the datetime part. The ANYDTTME informat extracts the time part.

Example

```
input dateinfo anydtdtm21.;
```

Data Line	Informat or Form of Data	Result	Formatted with DATETIME *w.d* Format
`----+----1----+----2`			
`01JAN2012`	DATE	`1640995200`	`01JAN12:00:00:00`
`01JAN2012 14:30:08.5`	DATETIME	`1641047408.5`	`01JAN12:14:30:09`
`01012012`	DDMMYY	`1640995200`	`01JAN12:00:00:00`
`2012001`	JULIAN	`1546387200`	`01JAN12:00:00:00`
`01/01/12`	MMDDYY	`1546387200`	`01JAN12:00:00:00`
`01-12`	MMxYY	`1546387200`	`01JAN12:00:00:00`
`JAN2012`	MONYY	`1546387200`	`01JAN12:00:00:00`
`14:30`	TIME	`52200`	`01JAN60:14:30:00`
`20120101`	YYMMDD	`1546387200`	`01JAN12:00:00:00`
`12Q1`	YYQ	`1546387200`	`01JAN12:00:00:00`
`January 1, 2012`	*month-day-year*	`1546387200`	`01JAN12:00:00:00`

See Also

Informats:

- "ANYDTDTEw. Informat" on page 242
- "ANYDTTMEw. Informat" on page 248
- "DATEw. Informat" on page 267
- "DATETIMEw. Informat" on page 268
- "DDMMYYw. Informat" on page 270
- "JULIANw. Informat" on page 289
- "MMDDYYw. Informat" on page 292
- "MONYYw. Informat" on page 294

- "TIMEw. Informat" on page 332
- "YYMMDDw. Informat" on page 347
- "YYQw. Informat" on page 350

ANYDTTME*w*. Informat

Reads and extracts time values from various date, time, and datetime forms.

Category: Date and Time

Syntax

ANYDTTME*w*.

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 1–32

Details

The ANYDTTME informat reads input data that corresponds to any of the following informats or forms.

Informat or Form of Input	Example Data	Informat or Form of Input	Example Data
DATE	01JAN12 01JAN2012	MONYY	JAN12 JAN2012
DATETIME	01JAN12 14:30:08 01JAN2012 14:30:08.5	YYMMDD	120101 20120101
DDMMYY	010112 01012012	YYQ	12Q1 2012Q1
JULIAN	12001 2012001	YYQ	12Q1 2012Q1
MMDDYY	010112 01012012	*month-day-year*	January 1, 2012 2012-01

If the input value is a time-only value, then SAS assumes a date of 01JAN1960. If the input value is a date value only, then SAS assumes a time of 12:00 midnight.

It is possible for input data such as 01-02-03 or 01-02 to be ambiguous with respect to the month, day, and year. In this case, the DATESTYLE system option indicates the order of the month, day, and year.

The ANYDTTME informat uses the following rules when reading colons and periods in time values:

Use of Colons and Periods	Example
a single colon in the value *h*:*m* indicates hours and minutes	14:30
two colons in the value h:m:s indicate hours, minutes, and seconds	14:30:08
a single period in the value m:s.ff, where ff is a fraction of a second, indicates that the number preceding the period is the number of seconds	2:39.66
multiple periods in the value indicate that the period is a delimiter for dates and the value is not a time value.	12.25.2012

Comparisons

The ANYDTDTE informat extracts the date part from the derived value. The ANYDTDTM informat extracts the datetime part. The ANYDTTME informat extracts the time part.

Example

```
input dateinfo anydttme21.;
```

Data Line	Informat	Result	Formatted with the TIME*w.d* Format
`----+----1----+----2`			
`01JAN12`	DATE	`0`	`00:00:00`
`01JAN2012 14:30:08.5`	DATETIME	`52208.5`	`14:30:09`
`010112`	DDMMYY	`0`	`00:00:00`
`2012001`	JULIAN	`0`	`00:00:00`
`01012012`	MMDDYY	`0`	`00:00:00`
`JAN2012`	MONYY	`0`	`00:00:00`
`14:30:08.5`	TIME	`52208.5`	`14:30:09`

Data Line	Informat	Result	Formatted with the TIME*w.d* Format
20120101	YYMMDD	0	00:00:00
12Q1	YYQ	0	00:00:00
January 1, 2012	*month-day-year*	0	00:00:00

See Also

Informats:

- "ANYDTDTEw. Informat" on page 242
- "ANYDTDTMw. Informat" on page 245
- "DATEw. Informat" on page 267
- "DATETIMEw. Informat" on page 268
- "DDMMYYw. Informat" on page 270
- "JULIANw. Informat" on page 289
- "MMDDYYw. Informat" on page 292
- "MONYYw. Informat" on page 294
- "TIMEw. Informat" on page 332
- "YYMMDDw. Informat" on page 347
- "YYQw. Informat" on page 350

B8601CI*w.d* Informat

Reads an IBM date and time value that includes a century marker, in the form *cyymmddhhmmss<fff>*.

Categories: Date and Time
ISO 8601

Alignment: left

Syntax

B8601CI*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 16

Range: 10–26

d

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

Default: 0

Range: 0–6

Details

The B8601CI informat reads time values that are specified in the IBM time notation *cyymmddhhmmss<fff>*:

c

is a single digit that represents a century:

0 indicates the years 1900–1999.

1 indicates the years 2000–2099.

2 indicates the years 2100–2199.

n indicates the years 00–99 in a century that is determined by performing a calculation on a year greater than 2199. To determine the century marker, subtract 1900 from the year and divide the result by 100. Discard the remainder. The remaining integer is the century marker. For example, to determine the century marker for the year 2382, perform this calculation: (2382–1900)/100=4.82. Discard .82. The century marker is 4.

yy

is a two-digit year between 00 and 99.

mm

is a two-digit month (zero padded) between 01 and 12.

dd

is a two-digit day of the month (zero padded) between 01 and 31.

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

fff

are optional fractional seconds, with a precision of up to three digits, where each digit is between 0 and 9.

Example

```
input @1 bci b8601ci.;
```

Date and Time	Data Line	Result
	`----+----1----+`	
January 1, 1900 12:00:00	`0000101120000`	`-1893326400`
October 1, 2011 12:15:30.25	`111100112153025`	`1633090530.3`

B8601DA*w*. Informat

Reads date values that are specified using the ISO 8601 base notation *yyyymmdd*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: ND8601DA

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

B8601DA*w*.

Syntax Description

w
specifies the width of the input field.

Default: 10

Requirement: The width of the output field must be 10.

Details

The B8601DA informat reads date values that are specified using the ISO 8601 basic date notation *yyyymmdd*:

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

If the month or day values are omitted, SAS uses a value of 1 for the month or day. If the hour, minute, or second values are omitted, SAS uses a value of 0 for the hour, minute, or second.

Example

```
input @1 bda b8601da.;
```

Data Line	Result	Formatted Using B8601DA Format
----+----1		
20120915	19251	20120915

Data Line	Result	Formatted Using B8601DA Format
2012	18993	20120101

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

B8601DJ*w.d* Informat

Reads a Java date and time value that is in the form *yyyymmddhhmmss<ffffff>*.

Categories: Date and Time
ISO 8601

Alignment: left

Syntax

B8601DJ*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 16

Range: 10–26

d

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

Default: 0

Range: 0–6

Details

The B8601DJ informat reads a date and time value that is specified using the Java date and time notation *yyyymmddhhmmss<ffffff>*:

yyyy
is a four-digit year between 0000 and 9999.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

hh
is a two-digit hour (zero padded) between 00 and 23.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

ffffff
: are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Comparisons

The B8601DJ informat reads a date and time value that does not include a T to separate the date from the time.

Java date and time values do not include a T. For example, the date January 1, 2011 at 4:30:25 a.m. is written as 20110101043025.

ISO 8601 date and time values include a T. For example, the date January 1, 2011 at 4:30:25 a.m. is written as 20110101T043025.

Example

```
input @1 bdj b8601dj.;
```

Date and Time	Data Line	Result
	`----+----1----+-----`	
July 31, 2011 2:33:35 p.m.	`20110731142335`	`1627741415`
September 1, 2012 7:30:00.33 a.m.	`2012090107300033`	`1662103800.3`

B8601DN*w*. Informat

Reads date values that are specified using the ISO 8601 basic notation *yyyymmdd* and returns SAS datetime values where the time portion of the value is 000000.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: ND8601DN

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

B8601DN*w*.

Syntax Description

w
: specifies the width of the input field.

Default: 10

Requirement: The width of the input field must be 10.

Details

The B8601DN informat reads date values that are specified using the ISO 8601 basic date notation *yyyymmdd* and returns the date in a SAS datetime value:

yyyy
: is a four-digit year.

mm
: is a two-digit month (zero padded) between 01 and 12.

dd
: is a two-digit day of the month (zero padded) between 01 and 31.

Example

```
input @1 bdn b8601dn.;
```

Data Line	Result
`----+----1`	
`20120915`	`1663286400`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

B8601DT*w.d* Informat

Reads datetime values that are specified using the ISO 8601 basic notation *yyyymmdd*T*hhmmss*<*ffffff*>.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: ND8601DT

Restriction: UTC time zone offset values are not supported.

Syntax

B8601DT*w.d*

Syntax Description

w
: specifies the width of the input field.

Default: 19

Range: 19–26

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–6

Details

The B8601DT informat reads datetime values that are specified in the ISO 8601 basic datetime notation *yyyymmdd*T*hhmmss*<*ffffff*>:

yyyy
: is a four-digit year.

mm
: is a two-digit month (zero padded) between 01 and 12.

dd
: is a two-digit day of the month (zero padded) between 01 and 31.

hh
: is a two-digit hour (zero padded) between 00 and 23.

mm
: is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

ffffff
: are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

If the month or day values are omitted, SAS uses a value of 1 for the month or day. If the hour, minute, or second values are omitted, SAS uses a value of 0 for the hour, minute, or second.

Example

```
input @1 bdt b8601dt.;
```

Data Line	Result	Formatted Using B8601DT Format
`----+----1`		
`20120915T155300`	`1663343580`	`20120915T155300`
`2012`	`1640995200`	`20120101T000000`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

B8601DZ*w.d* Informat

Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation *yyyymmdd*T*hhmmss*+|–*hhmm* or *yyyymmdd*T*hhmmss*<*ffffff*>Z.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: ND8601DZ

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

B8601DZ*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 26

Range: 20–35

d
specifies the number of digits to the right of the seconds value, which represents a fraction of a second. This argument is optional.

Default: 0

Range: 0–6

Details

UTC values specify a time and a time zone based on the zero meridian in Greenwich, England. The B8601DZ informat reads datetime values that are specified in one of the following ISO 8601 basic datetime notations:

- *yyyymmdd*T*hhmmss*+|–*hhmm*
- *yyyymmdd*T*hhmmss*<*ffffff*>Z

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
is a two-digit day of the month (zero padded) between 01 and 31.

hh
is a two-digit hour (zero padded) between 00 and 24.

mm
is a two-digit minute (zero padded) between 00 and 59.

ss
is a two-digit second (zero padded) between 00 and 59.

ffffff
are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

+|–*hhmm*
is an hour and minute signed offset from zero meridian time. Note that the offset must be +|–*hhmm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six–hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Z
indicates that the time is for zero meridian (Greenwich, England) or +0000 UTC time.

Example

```
input @1 bdz b8601dz.;
```

Data Line	Result
`----+----1`	
`20120915T155300+0500`	`1663325580`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

B8601TM*w.d* Informat

Reads time values that are specified using the ISO 8601 basic notation *hhmmss<ffffff>*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: ND8601TM

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Elements 5.3.1.1 and 5.3.1.3, complete representation and representation of decimal fractions

Syntax

B8601TM*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 6–15

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–6

Details

The B8601TM informat reads time values that are specified using the ISO 8601 basic time notation *hhmmss<ffffff>*:

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

ffffff

are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```
input @1 btm b8601tm;
```

Data Line	Result
`----+----1`	
`155300`	`57180`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

B8601TZ*w.d* Informat

Reads time values that are specified using the ISO 8601 basic time notation *hhmmss<ffff>+|–hhmm* or *hhmmss<ffffff>*Z.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: ND8601TZ

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.3.1.1, complete representation

Syntax

B8601TZ*w*.*d*

Syntax Description

w

specifies the width of the input field.

Default: 14

Range: 9–20

d

(optional) specifies the number of digits to the right of the decimal point in the seconds value.

Default: 0

Range: 0–6

Details

UTC time values specify a time and a time zone based on the zero meridian in Greenwich, England. The B8601TZ informat reads time values that are specified in the following ISO 8601 basic time notations:

- *hhmmss*<*ffffff*>+|–*hhmm*
- *hhmmss*<*ffffff*>Z

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

ffffff

are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

+|–*hh:mm*

is an hour and minute signed offset from zero meridian time. Note that the offset must be +|–*hhmm* (that is, + or – and four characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Z

indicates that the time is for zero meridian (Greenwich, England) or +0000 UTC time.

When SAS reads a UTC time by using the B8601TZ informat and the adjusted time is greater than 240000 or less than 000000, SAS adjusts the time value so that the time is between 000000 and 240000. For example, if SAS reads the UTC time 234344–0500 using the B8601TZ informat, SAS adds five hours to the time so that the value is 284344, and then makes the time adjustment. The value stored represents the time 044344+0000.

Example

```
input @1 btz b8601tz.;
```

Data Line	Result
`----+----1`	
`202401-0500`	`5041`
`202401Z`	`73441`
`202401+0000`	`73441`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

BINARY*w.d* Informat

Converts positive binary values to integers.

Category: Numeric

Syntax

BINARY*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 1–64

d

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

Range: 0–31

Details

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

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

Example

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

Data Line	Result
----+----1	
00001111	1.5

BITS*w.d* Informat

Extracts bits.

Category: Numeric

Syntax

BITS*w.d*

Syntax Description

w

specifies the number of bits to read.

Default: 1

Range: 1–64

d

specifies the zero-based offset.

Range: 0–63

Details

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

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

Example

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

Data Line	Result *
----+----1----+	

Data Line	Result *
B	8

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

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

BZ*w.d* Informat

Converts blanks to 0s.

Category: Numeric

Syntax

BZ*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–32

d

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

Range: 0–31

Details

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

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

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

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

Comparisons

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

Example

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

Data Line	Result
----+----1	
34	3400
-2	-200
-2 1	-201

CB*w.d* Informat

Reads standard numeric values from column-binary files.

Category: Column Binary

Syntax

CB*w*.*d*

Syntax Description

w

specifies the width of the input field.

Range: 1–32

d

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

Details

Column-binary data storage compresses data so that more than 80 items of data can be stored on a single "virtual" punch card.

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

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

Example: Examples

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

Data Line *	Result
`----+----1`	
`0009`	9

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

See Also

- “How to Read Column-Binary Data” in Chapter 19 of *SAS Language Reference: Concepts* in *SAS Language Reference: Concepts*

Informats

- “$CBw. Informat” on page 227
- “PUNCH.d Informat” on page 308
- “ROWw.d Informat” on page 313

COMMA*w.d* Informat

Removes embedded characters.

Category: Numeric

Alias: DOLLAR*w.d*

Syntax

COMMA*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–32

d

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

Range: 0–31

Details

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

Comparisons

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

Example

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

Data Line	Result
`----+----1----+`	
`$1,000,000`	`1000000`
`(500)`	`-500`

COMMAX*w.d* Informat

Removes embedded periods, blanks, dollar signs, percent signs, dashes, and closing parenthesis from the input data. An open parenthesis at the beginning of a field is converted to a minus sign. The COMMAX informat reverses the roles of the decimal point and the comma.

Category: Numeric

Alias: DOLLARX*w.d*

Syntax

COMMAX*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–32

d

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

Range: 0–31

Details

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

Comparisons

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

Example

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

Data Line	Result
`----+----1----+`	
`$1.000.000`	`1000000`
`1.234,56`	`1234.56`
`(500)`	`-500`

DATE*w*. Informat

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

Category: Date and Time

Syntax

DATE*w*.

Syntax Description

w

specifies the width of the input field.

Default: 7

Range: 7–32

Tip: Use a width of 9 to read a 4–digit year.

Details

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

dd
: is an integer between 01 and 31 that represents the day of the month.

mmm
: is the first three letters of the month name.

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

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

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

Example

```
input calendar_date date11.;
```

Data Line	Result
----+----1----+	
16mar12	19068
16 mar 12	19068
16-mar-2012	19068

See Also

Formats:

- "DATEw. Format" on page 73

Functions:

- "DATE Function" in *SAS Functions and CALL Routines: Reference*

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

DATETIME*w*. Informat

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

Category: Date and Time

Syntax

DATETIME*w*.

Syntax Description

w

specifies the width of the input field.

Default: 18

Range: 13–40

Details

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

dd
: is an integer between 01 and 31 that represents the day of the month.

mmm
: is the first three letters of the month name.

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

hh
: is an integer between 00 and 23 that represents hours.

mm
: is an integer between 00 and 59 that represents minutes.

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

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

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

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

Comparisons

The DATETIME*w.d* informat reads datetime values with optional separators in the form *dd-mmm-yy<yy> hh:mm:ss.ss* AM|PM, and the date and time can be separated by a special character.

The MDYAMPM*w.d* in format reads datetime values with optional separators in the form *mm-dd-yy<yy> hh:mm:ss.ss* AM | PM, and requires a space between the date and the time.

The YMDDTTM*w.d* informat reads datetime values with required separators in the form *<yy>yy-mm-dd/hh:mm:ss.ss*.

Example

```
input date_and_time datetime20.;
```

Data Line	Result
`----+----1----+----2`	
`16mar12:11:23:07.4`	`1647516187.4`
`16mar2012/11:23:07.4`	`1647516187.4`
`16mar2012/11:23 PM`	`1647559380.0`

See Also

- "About SAS Date, Time, and Datetime Values" in Chapter 7 of *SAS Language Reference: Concepts*

Formats:

- "DATEw. Format" on page 73
- "DATETIMEw.d Format" on page 75
- "TIMEw.d Format" on page 157

Functions:

- "DATETIME Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267
- "MDYAMPMw.d Informat" on page 291
- "TIMEw. Informat" on page 332
- "YMDDTTMw.d Informat" on page 345

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

DDMMYY*w*. Informat

Reads date values in the form *ddmmyy<yy>* or *dd-mm-yy<yy>*, where a special character, such as a hyphen (-), period (.), or slash (/), separates the day, month, and year; the year can be either 2 or 4 digits.

Category: Date and Time

Syntax

DDMMYY*w*.

Syntax Description

w

specifies the width of the input field.

Default: 6

Range: 6–32

Details

The date values must be in the form *ddmmyy<yy>* or *ddxmmxyy<yy>*:

dd

is an integer between 01 and 31 that represents the day of the month.

mm

is an integer between 01 and 12 that represents the month.

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

x
is a separators that can be any special character or a blank.:

If you use separators, place them between all the values. Blanks can also be placed before and after the date. Make sure the width of the input field allows space for blanks and special characters.

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

Example

```
input calendar_date ddmmyy10.;
```

Data Line	Result
----+----1----+	
160308	19068
16/03/08	19068
16-03-2008	19068
16 03 2008	19068

See Also

Formats:

- “DATEw. Format” on page 73
- “DDMMYYw. Format” on page 78
- “MMDDYYw. Format” on page 113
- “YYMMDDw. Format” on page 181

Functions:

- “MDY Function” in *SAS Functions and CALL Routines: Reference*

Informats:

- “DATEw. Informat” on page 267
- “MMDDYYw. Informat” on page 292
- “YYMMDDw. Informat” on page 347

System Options:

- “YEARCUTOFF= System Option” in *SAS System Options: Reference*

E*w.d* Informat

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

Category: Numeric

See: E*w.d* Informat under z/OS

Syntax

E*w.d*

Syntax Description

w

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

Default: 12

Range: 1–32

d

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

Range: 0–31

Comparisons

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

Example

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

Data Line	Result
----+----1----+	
1.257E3	1257
12d3	12000

E8601DA*w.* Informat

Reads date values that are specified using the ISO 8601 extended notation *yyyy-mm-dd*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601DA

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

E8601DA*w*.

Syntax Description

w

specifies the width of the input field.

Default: 10

Requirement: The width of the input field must be 10.

Details

The E8601DA informat reads date values that are specified in the ISO 8601 extended date notation *yyyy-mm-dd*:

yyyy

is a four-digit year.

mm

is a two-digit month (zero padded) between 01 and 12.

dd

is a two-digit day of the month (zero padded) between 01 and 31.

Example

```
input eda e8601da.;
```

Data Line	Result
`----+----1`	
`2012-09-15`	`19251`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

E8601DN*w*. Informat

Reads date values that are specified using the ISO 8601 extended notation *yyyy-mm-dd* and returns SAS datetime values where the time portion of the value is 000000.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601DN

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

E8601DN*w*.

Syntax Description

w

specifies the width of the input field.

Default: 10

Requirement: The width of the input field must be 10.

Details

The E8601DN informat reads date values that are specified using the ISO 8601 extended date notation *yyyy-mm-dd* and returns the date in a SAS datetime value:

yyyy

is a four-digit year.

mm

is a two-digit month (zero padded) between 01 and 12.

dd

is a two-digit day of the month (zero padded) between 01 and 31.

Example

```
input edn e8601dn.;
```

Data Line	Result
----+----1	
2012-09-15	1663286400

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

E8601DT*w.d* Informat

Reads datetime values that are specified using the ISO 8601 extended notation *yyyy-mm-dd*T*hh:mm:ss.<ffffff>*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601DT

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

E8601DT*w.d*

Syntax Description

w
: specifies the width of the input field.

Default: 19

Range: 19–26

d
: specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–6

Details

The E8601DT informat reads datetime values that are specified using the ISO 8601 extended datetime notation *yyyy-mm-dd*T*hh:mm:ss.*<*ffffff*>:

yyyy
: is a four-digit year.

mm
: is a two-digit month (zero padded) between 01 and 12.

dd
: is a two-digit day of the month (zero padded) between 01 and 31.

hh
: is a two-digit hour (zero padded) between 00 and 23.

mm
: is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

ffffff
: are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```
input @1 edt e8601dt.;
```

Data Line	Result
`----+----1----+----2----+----3`	

Data Line	Result
2012-09-15T15:53:00	1663343580

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

E8601DZ*w.d* Informat

Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime extended notation *yyyy-mm-dd*T*hh*:*mm*:*ss*+|–*hh*:*mm*.<*fffff*> or *yyyy-mm-dd*T*hh*:*mm*:*ss*.<*fffff*>Z.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601DZ

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

E8601DZ*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 26

Range: 20–35

d

specifies the number of digits to the right of the decimal point in the value for the lowest-order component. This argument is optional.

Default: 0

Range: 0–6

Details

UTC values specify a time and a time zone based on the zero meridian in Greenwich, England. The E8601DZ informat reads datetime values that contain UTC time offsets and that are specified in one of the following ISO 8601 extended datetime notations:

- *yyyy-mm-dd*T*hh:mm:ss*.<*ffffff*>+|–*hh:mm*
- *yyyy-mm-dd*T*hh:mm:ss*.<*ffffff*>Z

yyyy
is a four-digit year.

mm
is a two-digit month (zero padded) between 01 and 12.

dd
: is a two-digit day of the month (zero padded) between 01 and 31.

hh
: is a two-digit hour (zero padded) between 00 and 24.

mm
: is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

ffffff
: are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

+|–*hh:mm*
: is an hour and minute signed offset from zero meridian time. Note that the offset must be +|–*hh:mm* (that is, + or – and five characters).

Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +02:00 indicates a two-hour time difference to the east of the zero meridian, and –06:00 indicates a six-hour time difference to the west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Z
: indicates that the time is UTC time at the zero meridian (Greenwich, England).

Example

Input Statement	Data Line	Result
	`----1----+----2----+----3`	
`input edz e8601dz.;`	`2012-09-15T15:53:00Z`	`1663343580`
`input edz  e8601dz28.2;`	`2012-09-15T15:53:00+03:00`	`1663332780`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

E8601LZ*w.d* Informat

Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation *hh*:*mm*:*ss*+|–*hh*:*mm*.<*fffff*> or *hh*:*mm*:*ss*.<*fffff*>Z and converts the values to the local time.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601LZ

Supports: ISO 8601 Element 5.3.1.1, complete representation

Syntax

E8601LZ*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 14

Range: 9–20

Requirement: To read a time with the Z time zone indicator, the width of the input field must be 9 if data follows on the same line of data.

d

specifies the number of digits to the right of the decimal point in the value for the lowest-order component. This argument is optional.

Default: 0

Range: 0–6

Details

UTC values specify a time and a time zone based on the zero meridian in Greenwich, England. The E8601LZ informat reads UTC time values that are specified in one of the following ISO 8601 extended time notations and returns a SAS time value for the local time:

- *hh:mm:ss.<ffffff>+|–hh:mm*
- *hh:mm:ss.<ffffff>*Z

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

ffffff

are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

+|–*hh:mm*

is an hour and minute signed offset from zero meridian. Note that the offset must be +|–*hh:mm* (that is, + or – and five characters).

Use the + for time zones east of the zero meridian, and use the – for time zones west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Z

indicates zero meridian or +00:00 UTC time.

When SAS reads a UTC time by using the E8601LZ informat and the adjusted time is greater than 24:00:00 or less than 00:00:00, SAS adjusts the value so that the time is between 00:00:00 and 24:00:00. For example, if SAS reads the UTC time 23:43:44-05:00 by using the E8601LZ informat, SAS adds five hours to the time so that the value is 28:43:44, and then makes the time adjustment. The value stored represents the time 04:43:44+00:00.

Example

```
input elz e8601lz.;
```

Data Line	Result
`----+----1----+`	
`09:13:21+02:00`	`26001`
`23:43:44Z`	`85424`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

E8601TM*w.d* Informat

Reads time values that are specified using the ISO 8601 extended notation *hh:mm:ss.<ffffff>*.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601TM

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Elements 5.3.1.1 and 5.3.1.3, complete representation and representation of decimal fractions

Syntax

E8601TM*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 8–15

d

specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

Default: 0

Range: 0–6

Details

The E8601TM informat reads time values that are specified using the ISO 8601 extended time notation *hh:mm:ss.<ffffff>*:

hh
: is a two-digit hour (zero padded) between 00 and 23.

mm
: is a two-digit minute (zero padded) between 00 and 59.

ss
: is a two-digit second (zero padded) between 00 and 59.

ffffff
: are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```
input @1 etm e8601tm.;
```

Data Line	Result
`----+----1`	
`15:53:00`	`57180`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

E8601TZ*w.d* Informat

Reads time values that are specified using the ISO 8601 extended time notation *hh*:*mm*:*ss*+|–*hh*:*mm*.<*ffffff*> or *hh*:*mm*:*ss*Z.

Categories: Date and Time
ISO 8601

Alignment: left

Alias: IS8601TZ

Supports: ISO 8601 Element 5.3.1.1, complete representation

Syntax

E8601TZ*w.d*

Syntax Description

w
: specifies the width of the input field.
Default: 14

Range: 9–20

Requirement: To read a time with the Z time zone indicator, the width of the input field must be 9 if data follows on the same line of data.

d

specifies the number of digits to the right of the decimal point in the value for the lowest-order component. This argument is optional.

Default: 0

Range: 0–6

Details

UTC time values specify a time and a time zone based on the zero meridian in Greenwich, England. The E8601TZ informat reads UTC time values that are specified in one of the following ISO 8601 extended notations:

- *hh:mm:ss*+|–*hh:mm*.<*ffffff*>
- *hh:mm:ss*Z

hh

is a two-digit hour (zero padded) between 00 and 23.

mm

is a two-digit minute (zero padded) between 00 and 59.

ss

is a two-digit second (zero padded) between 00 and 59.

ffffff

are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

+|–*hh:mm*

is an hour and minute signed offset from zero meridian. Note that the offset must be +|–*hh:mm* (that is, + or – and five characters).

Use the + for time zones east of the zero meridian, and use the – for time zones west of the zero meridian.

Restriction: The shorter form +|–*hh* is not supported.

Z

indicates zero meridian or +00:00 UTC time.

When SAS reads a UTC time by using the E8601TZ informat and the adjusted time is greater than 24:00:00 or less than 00:00:00, SAS adjusts the value so that the time is between 00:00:00 and 24:00:00. For example, if SAS reads the UTC time 23:43:44–05:00 by using the E8601TZ informat, SAS adds five hours to the time so that the value is 28:43:44, and then makes the time adjustment. The value stored represents the time 04:43:44+00:00.

Example

```
input @1 etz e8601tz.;
```

Data Line	Result
`----+----1----+`	

Data Line	Result
`23:43:44-05:00`	`17024`
`23:43:44Z`	`85424`

See Also

"Reading Dates and Times by Using the ISO 860 Basic and Extended Notations" on page 209

FLOAT*w.d* Informat

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

Category: Numeric

Syntax

FLOAT*w.d*

Syntax Description

w

specifies the width of the input field.

Requirement: *w* must be 4.

d

specifies the power of 10 by which to divide the value. This argument is optional.

Details

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

On the IBM mainframe systems, a four-byte floating-point number is the same as a truncated eight-byte floating-point number. However, in operating environments that use the IEEE floating-point standard, such as the IBM PC-based operating environments and most UNIX platforms, a four-byte floating-point number is not the same as a truncated double. Therefore, the RB4. informat does not produce the same results as FLOAT4. Floating-point representations other than IEEE might have this same characteristic. Values read with FLOAT4. typically come from some other external program that is running in your operating environment.

Comparisons

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

Language	Float Notation
SAS	FLOAT4.

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

Example

```
input x float4.;
```

Data Line *	Result
`----+----1----+----2`	
`3F800000`	`1`

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

HEX*w*. Informat

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

Category: Numeric

See: "HEXw. Informat: UNIX" in *SAS Companion for UNIX Environments*
"HEXw. Informat: Windows" in *SAS Companion for Windows*
"HEXw. Informat: z/OS" in *SAS Companion for z/OS*

Syntax

HEX*w*.

Syntax Description

w

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

Default: 8

Range: 1–16

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

Details

Operating Environment Information

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

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

Example

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

Data Line *	Result
`----+----1----+----2`	
`88F 4152000000000000`	`2191 5.125`

* The data line shows IBM mainframe hexadecimal data.

HHMMSS*w*. Informat

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

Category: Date and Time

Syntax

HHMMSS*w*.

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 1–20

Details

The HHMMSS*w*. informat reads SAS time values in one of the following forms:

- *hh*:*mm*:*ss*
- *hhmmss*

hh

is an integer that represents the number of hours.

:

represents a special character that separates hours, minutes, and seconds.

mm

is an integer that represents the number of minutes.

ss
: is an integer that represents the number of seconds. Fractional seconds are ignored.

If the input data is six digits or less, SAS reads the data from left to right as hours, minutes, and seconds. Any data less than six digits is padded to the right with zeros. The first two digits are read as hours. Digits three and four are read as minutes. Digits five and six are read as seconds.

1 is the same as 100000 or 10:00:00.

02 is the same as 020000 or 02:00:00.

124 is the same as 124000 or 12:40:00.

1435 is the same as 143500 or 14:35:00.

20345 is the same as 203450 or 20:34:50.

165532 is the same as 16:55:32.

When there are more than six digits, SAS reads the last two digits from the right as seconds. The third and forth digits from the right are read as minutes. The remaining digits to the left of the minutes are read as hours.

2358444 is the same as 235:84:44.

12545533 is the same as 1254:55:33.

If the input data has only one colon (for example, 17:35), the two digits before the colon are read as hours. The two digits after the colon are read as seconds. The number of seconds is 0.

Note: If a colon is omitted between minutes and seconds, as in 12:3400, the 3400 is read as 3400 minutes. 3400 minutes adds 56 hours and 40 minutes to the 12 hours, resulting in 68:40:00. See the following example.

Example

input tm hhmmss.;

Data Line	Result	Formatted with TIME*w*.
`23`	`82800`	`23:00:00`
`12:45:44`	`45344`	`12:45:44`
`2358444`	`851084`	`236:24:44`
`17:35`	`63300`	`17:35:00`
`12:3400`	`247200`	`68:40:00`

See Also

Informats:

- “TIMEw. Informat” on page 332

IB*w.d* Informat

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

Category: Numeric

See: “IBw.d Informat: UNIX” in *SAS Companion for UNIX Environments*
“IBw.d Informat: Windows” in *SAS Companion for Windows*
“IBw.d Informat: z/OS” in *SAS Companion for z/OS*

Syntax

IB*w.d*

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 1–8

d
specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 203 .

Comparisons

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

To view a table that shows the type of informat to use with big endian and little endian integers, see Table 3.1 on page 204.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 9.

Example

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

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

SAS Statement	Result on Big Endian Platforms	Result on Little Endian Platforms
`put x=;`	128	-32768
`put y=;`	-32768	128

See Also

Informats:

- "IBRw.d Informat" on page 287

IBR*w.d* Informat

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

Category: Numeric

Syntax

IBR*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 4

Range: 1–8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms" on page 203 .

Comparisons

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

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

To view a table that shows the type of informat to use with big endian and little endian integers, see Table 3.1 on page 204.

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

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

SAS Statement	Result on BigEndian Platforms	Result on LittleEndian Platforms
put x=;	1	1
put y=;	256	256

See Also

Informats:

- "IBw.d Informat" on page 286

IEEE*w.d* Informat

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

Category: Numeric

Syntax

IEEE*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 2–8

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

d

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

Details

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

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

Example

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

Data Line *	Result
----+----1----+	
3F800000	1
3FF0000000	1

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

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

JULIAN*w.* Informat

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

Category: Date and Time

Syntax

JULIAN*w*.

Syntax Description

w

specifies the width of the input field.

Default: 5

Range: 5–32

Details

The date values must be in one of the following forms:

- *yyddd*
- *yyyyddd*

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

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

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

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

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

Example

```
input julian_date julian7.;
```

Data Line	Result *
----+----1	
12076	19068
2012076	19068

* The input values correspond to the 76th day of 2012, which is March 16.

See Also

Formats:

- "JULIANw. Format" on page 111

Functions:

- "DATEJUL Function" in *SAS Functions and CALL Routines: Reference*
- "JULDATE Function" in *SAS Functions and CALL Routines: Reference*

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

MDYAMPM*w.d* Informat

Reads datetime values in the form *mm-dd-yy<yy> hh:mm:ss.ss* AM|PM, where a special character such as a hyphen (-), period (.), slash (/), or colon (:) separates the month, day, and year; the year can be either 2 or 4 digits.

Category: Date and Time

Alignment: right

Requirement: A space must separate the date and the time.

Note: The default time period is AM.

Syntax

MDYAMPM*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 19

Range: 8–40

d

specifies the number of digits to the right of the decimal point in the seconds value. The digits to the right of the decimal point specify a fraction of a second. This argument is optional.

Default: 0

Range: 0–39

Details

The MDYAMPMw.d format reads SAS datetime values in the form *mm-dd-yy<yy> hh*:*mm*<:*ss*<.*ss*>> <AM | PM>:

mm

is an integer between 01 and 12 that represents the month.

dd

is an integer between 01 and 31 that represents the day of the month.

yy or *yyyy*

specifies a two-digit or four-digit integer that represents the year.

hh

is an integer between 00 and 23 that represents hours.

mm

is an integer between 00 and 59 that represents minutes.

ss.ss
is the number of seconds that range from 00–59 with the fraction of a second following the decimal point.

Requirement: If a fraction of a second is specified, the decimal point can be represented only by a period and is required.

AM | PM
specifies either the time period 00:01–12:00 noon (AM) or the time period 12:01–12:00 midnight (PM)

- or :
represents one of several special characters, such as the slash (/), hyphen (-), colon (:), or a blank character that can be used to separate date and time components. Special characters can be used as separators between any date or time component and between the date and the time.

Comparisons

The MDYAMPM*w.d* informat reads datetime values with optional separators in the form *mm-dd-yy<yy> hh:mm:ss.ss* AM | PM, and requires a space between the date and the time.

The DATETIME*w.d* informat reads datetime values with optional separators in the form *dd-mmm-yy<yy> hh:mm:ss.ss* AM|PM, and the date and time can be separated by a special character.

The YMDDTTM*w.d* informat reads datetime values with required separators in the form *<yy>yy-mm-dd/hh:mm:ss.ss*.

Example

```
input @1 dt mdyampm25.2.;
```

Data Line	Result
`09.15.2012 03:53:00 pm`	`1663343580`
`09-15-12 3.53 pm`	`1663343580`

See Also

Informats:

- “DATETIMEw. Informat” on page 268
- “YMDDTTMw.d Informat” on page 345

MMDDYY*w*. Informat

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

Category: Date and Time

Syntax

MMDDYY*w*.

Syntax Description

w

specifies the width of the input field.

Default: 6

Range: 6–32

Details

The date values must be in one of the following forms:

- *mmddyy*
- *mmddyyyy*

mm

is an integer between 01 and 12 that represents the month.

dd

is an integer between 01 and 31 that represents the day of the month.

yy or *yyyy*

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

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

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

Example

```
input calendar_date mmddyy8.;
```

Data Line	Result
`----+----1----+`	
`031612`	`19068`
`03/16/12`	`19068`
`03 16 12`	`19068`
`03162012`	`19068`

See Also

Formats:

- "DATEw. Format" on page 73
- "DDMMYYw. Format" on page 78

- “MMDDYYw. Format” on page 113
- “YYMMDDw. Format” on page 181

Functions:

- “DAY Function” in *SAS Functions and CALL Routines: Reference*
- “MDY Function” in *SAS Functions and CALL Routines: Reference*
- “MONTH Function” in *SAS Functions and CALL Routines: Reference*
- “YEAR Function” in *SAS Functions and CALL Routines: Reference*

Informats:

- “DATEw. Informat” on page 267
- “DDMMYYw. Informat” on page 270
- “YYMMDDw. Informat” on page 347

System Options:

- “YEARCUTOFF= System Option” in *SAS System Options: Reference*

MONYY*w*. Informat

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

Category: Date and Time

Syntax

MONYY*w*.

Syntax Description

w
specifies the width of the input field.

Default: 5

Range: 5–32

Details

The date values must be in one of the following forms:

- *mmmyy*
- *mmmyyyy*

mmm
is the first three letters of the month name.

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

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

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

Example

```
input month_and_year monyy7.;
```

Data Line	Result
----+----1	
mar 12	19053
mar2012	19053

See Also

Formats:

- "DDMMYYw. Format" on page 78
- "MMDDYYw. Format" on page 113
- "MONYYw. Format" on page 123
- "YYMMDDw. Format" on page 181

Functions:

- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DDMMYYw. Informat" on page 270
- "MMDDYYw. Informat" on page 292
- "YYMMDDw. Informat" on page 347

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

MSEC*w*. Informat

Reads TIME MIC values.

Category: Date and Time

Syntax

MSEC*w*.

Syntax Description

w

specifies the width of the input field.

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

Details

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

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

Comparisons

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

Example

```
input btime msec8.;
```

Data Line *	Result
0000EA044E65A000	62818.412122

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

See Also

Informats:

- "TODSTAMPw. Informat" on page 334

NUMX*w.d* Informat

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

Category: Numeric

Syntax

NUMX*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 12

Range: 1–32

d

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

Range: 0–31

Details

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

Comparisons

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

Example

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

Data Line	Result
`----+----1----+`	
`896,48`	`896.48`
`3064,1`	`3064.1`
`6489`	`6489`

See Also

Formats:

- “NUMXw.d Format” on page 125
- “w.d Format” on page 164

OCTAL*w.d* Informat

Converts positive octal values to integers.

Category: Numeric

Syntax

OCTAL*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 3

Range: 1–24

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 1–31

Restriction: must be greater than or equal to the *w* value.

Details

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

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

Example

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

Data Line	Result
----+----1	
177	12.7

PD*w.d* Informat

Reads data that are stored in IBM packed decimal format.

Category: Numeric

See: "PDw.d Informat: UNIX" in *SAS Companion for UNIX Environments*
"PDw.d Informat: Windows" in *SAS Companion for Windows*
"PDw.d Informat: z/OS" in *SAS Companion for z/OS*

Syntax

PD*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–16

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

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

The PD*w.d* format writes missing numerical data as -0. When the PD*w.d* informat reads -0, it stores it as 0.

Comparisons

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

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

Examples

Example 1: Reading Packed Decimal Data

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

Data Line *	Result
----+----1	
0000128C	128

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

Example 2: Creating a SAS Date with Packed Decimal Data

```
input x: $hex10.;
mnth=input(x, pd5.);
date=input(put(mnth,8.),mmddyy6.);
```

Data Line *	Result
----+----1	

Data Line *	Result
012252010C	18621

* The data line is a hexadecimal representation of a binary number that is stored in packed decimal form on an IBM mainframe operating environment. Each byte occupies one column of the input field. The result is a SAS date value that corresponds to December 25, 2010.

PDJULG*w*. Informat

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

Category: Date and Time

Syntax

PDJULG*w*.

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 4

Details

The PDJULG*w*. informat reads IBM packed Julian date values in the form of *yyyyddd*F:

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.

Example

```
input date pdjulg4.;
```

Data Line	Result *
----+----1	
2012003F	18995

* SAS date value 18995 represents January 3, 2012.

See Also

Formats:

- "JULDAYw. Format" on page 110
- "JULIANw. Format" on page 111
- "PDJULGw. Format" on page 128
- "PDJULIw. Format" on page 130

Functions:

- "DATEJUL Function" in *SAS Functions and CALL Routines: Reference*
- "JULDATE Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "JULIANw. Informat" on page 289
- "PDJULIw. Informat" on page 301

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

PDJULI*w*. Informat

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

Category: Date and Time

Syntax

PDJULI*w*.

Syntax Description

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

Details

The PDJULI*w*. informat reads IBM packed Julian date values in the form *ccyyddd*F:

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

yy
is the one-byte representation of a two-digit integer that represents the year. The PDJULI*w* informat makes an adjustment to the one-byte century representation by adding 1900 to the two-byte *ccyy* value in order to produce the correct four–digit

Gregorian year. This adjustment causes *ccyy* values of 0098 to become 1998, 0101 to become 2001, and 0218 to become 2118.

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

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

Example

```
input date pdjuli4.;
```

Data Line	Result *
`----+----1`	
`0099001F`	`14245`
`0112015F`	`19007`

* SAS date value 14245 is January 1, 1999. SAS date value 19007 is January 15, 2012.

See Also

Formats:

- "JULDAYw. Format" on page 110
- "JULIANw. Format" on page 111
- "PDJULGw. Format" on page 128
- "PDJULIw. Format" on page 130

Functions:

- "DATEJUL Function" in *SAS Functions and CALL Routines: Reference*
- "JULDATE Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "JULIANw. Informat" on page 289
- "PDJULGw. Informat" on page 300

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

PDTIME*w*. Informat

Reads packed decimal time of SMF and RMF records.

Category: Date and Time

Syntax

PDTIME*w*.

Syntax Description

w
: specifies the width of the input field.

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

Details

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

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

0
: is a half byte that contains all 0s.

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

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

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

F
: is a half byte that contains all 1s.

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

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

Example

```
input begin pdtime4.;
```

Data Line *	Result
0142225F	51745

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

PERCENT*w.d* Informat

Reads percentages as numeric values.

Category: Numeric

Syntax

PERCENT*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 6

Range: 1–32

d

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

Range: 0–31

Details

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

Example

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

Data Line	Result
`----+----1`	
`1% (20%)`	`0.01 -0.2`

PIB*w.d* Informat

Reads positive integer binary (fixed-point) values.

Category: Numeric

See: "PIBw.d Informat: UNIX" in *SAS Companion for UNIX Environments*
"PIBw.d Informat: Windows" in *SAS Companion for Windows*

Syntax

PIB*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

Note: Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms" on page 203 .

Comparisons

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

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 3.1 on page 204.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

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

SAS Statement	Result on Big Endian Platforms	Result on Little Endian Platforms
`put x=;`	256	1
`put y=;`	1	256

See Also

Informats:

- "PIBRw.d Informat" on page 306

PIBR*w.d* Informat

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

Category: Numeric

Syntax

PIBR*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

Note: Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms" on page 203 .

Comparisons

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

environment.) The IBR*w.d* and PIBR*w.d* informats are used to read little endian integers in any operating environment.

To view a table that shows the type of informat to use with big endian and little endian integers, see Table 3.1 on page 204 .

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9.

Example

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

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

SAS Statement	Result on Big Endian Platforms	Result on Little Endian Platforms
`put x=;`	1	1
`put y=;`	256	256

See Also

Informat

- "PIBw.d Informat" on page 304

PK*w.d* Informat

Reads unsigned packed decimal data.

Category: Numeric

Syntax

PK*w.d*

Syntax Description

w

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

Default: 1

Range: 1–16

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

Each byte of unsigned packed decimal data contains two digits.

Comparisons

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

Example

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

Data Line *	Result
`----+----1`	
`001234`	`1234`

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

PUNCH.*d* Informat

Reads whether a row of column-binary data is punched.

Category: Column Binary

Syntax

PUNCH.*d*

Syntax Description

d

specifies which row in a card column to read.

Range: 1–12

Details

Column-binary data storage compresses data so that more than 80 items of data can be stored on a single “virtual” punch card.

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

Example

Data Line [*]	SAS Statement	Result
`12-7-8`	`input x punch.12`	`1`
	`input x punch.11`	`0`
	`input x punch0.7`	`1`

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

See Also

- "How to Read Column-Binary Data" in Chapter 19 of *SAS Language Reference: Concepts*

Informats:

- "$CBw. Informat" on page 227
- "CBw.d Informat" on page 264
- "ROWw.d Informat" on page 313

RB*w.d* Informat

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

Category: Numeric

See: "RBw.d Informat: UNIX" in *SAS Companion for UNIX Environments*
"RBw.d Informat: Windows" in *SAS Companion for Windows*
"RBw.d Informat: z/OS" in *SAS Companion for z/OS*

Syntax

RB*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 4

Range: 2–8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

Comparisons

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

	Real Binary Notation	
Language	**4 Bytes**	**8 Bytes**
SAS	RB4.	RB8.
Fortran	REAL*4	REAL*8
C	float	double
IBM 370 assembler	F	D
PL/I	FLOAT BIN(21)	FLOAT BIN(53)

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

Example

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

Data Line *	Result
----+----1	
4280000000000000	128

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

See Also

Informats:

- "IEEEw.d Informat" on page 288

RMFDUR*w*. Informat

Reads duration intervals of RMF records.

Category: Date and Time

Syntax

RMFDUR*w*.

Syntax Description

w

specifies the width of the input field.

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

Details

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

The general form of the duration interval data in an RMF record in hexadecimal notation is *mmsstttF*:

mm

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

ss

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

ttt

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

F

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

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

Comparisons

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

Example

```
input dura rmfdur4.;
```

Data Line *	Result
`----+----1`	
`3552226F`	`2152.226`

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

See Also

Informats:

- "RMFSTAMPw. Informat" on page 312
- "SMFSTAMPw. Informat" on page 330

RMFSTAMP*w*. Informat

Reads time and date fields of RMF records.

Category: Date and Time

Syntax

RMFSTAMP*w*.

Syntax Description

w

specifies the width of the input field.

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

Details

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

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

0

is the half byte that contains all binary 0s.

hh

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

mm

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

ss

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

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

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

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

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

The century indicators 00 correspond to 1900, 01 to 2000, and 02 to 2100.

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

Comparisons

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

Example

```
input begin: $hex16.;
y=input(begin, rmfstamp8.);
```

Data Line *	Result
`----+----1----+----2`	
`0142225F2612200F`	`80550512545`

* The data line is a hexadecimal representation of a binary time and date value that is stored in packed decimal form as it would appear in an RMF record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to July 18, 2012, 2:22.25 PM.

ROW*w.d* Informat

Reads a column-binary field down a card column.

Category: Column Binary

Syntax

ROW*w.d*

Syntax Description

w
: specifies the row where the field begins.

Range: 0–12

d

specifies the length in rows of the field.

Default: 1

Range: 1–25

Details

Column-binary data storage compresses data so that more than 80 items of data can be stored on a single “virtual” punch card.

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

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

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

Example

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

Data Line *	Result
`----+----1`	
`00`	
`04`	3
	1
	.
	5

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

See Also

- “How to Read Column-Binary Data” in Chapter 19 of *SAS Language Reference: Concepts*

Informats:

- “$CBw. Informat” on page 227
- “CBw.d Informat” on page 264
- “PUNCH.d Informat” on page 308

S370FF*w.d* Informat

Reads EBCDIC numeric data.

Category: Numeric

Syntax

S370FF*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 12

Range: 1–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–31

Details

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

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

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

Comparisons

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

Example

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

Data Line *	Result
----+----1	
F1F2F3	123

Data Line [*]	Result
F2F4F0	240

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

S370FIB*w.d* Informat

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

Category: Numeric

Syntax

S370FIB*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 4

Range: 1–8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

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

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 203 .

Comparisons

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

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 3.1 on page 204.

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9..

Example

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

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

SAS Statement	Result
put x=;	128

See Also

Informats

- "S370FIBUw.d Informat" on page 317
- "S370FPIBw.d Informat" on page 320

S370FIBU*w.d* Informat

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

Category: Numeric

Syntax

S370FIBU*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 4

Range: 1–8

d

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

Range: 0–10

Details

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

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

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see "Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms" on page 203 .

Comparisons

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

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 3.1 on page 204.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 9..

Example

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

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

SAS Statement	Result
`put x=;`	127
`put y=;`	246

See Also

Informats:

- "S370FIBw.d Informat" on page 316
- "S370FPIBw.d Informat" on page 320

S370FPD*w.d* Informat

Reads packed data in IBM mainframe format.

Category: Numeric

Syntax

S370FPD*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–16

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–31

Details

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

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

Comparisons

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

Language	Packed Decimal Notation
SAS	S370FPD4.
PL/I	FIXED DEC(7,0)
COBOL	COMP-3 PIC 9(7)
assembler	PL4

S370FPDU*w.d* Informat

Reads unsigned packed decimal data in IBM mainframe format.

Category: Numeric

Syntax

S370FPDU*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–16

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–31

Details

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

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

Comparisons

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

Example

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

Data Line *	Result
`----+----1`	
`12345F`	`12345`

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

S370FPIB*w.d* Informat

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

Category: Numeric

Syntax

S370FPIB*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 4

Range: 1–8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–10

Details

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

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

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 203 .

Comparisons

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

  To view a table that shows the type of informat to use with big endian and little endian integers, see Table 3.1 on page 204.

  To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 9.

Example

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

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

SAS Statement	Result
`put x=4;`	`256`

See Also

Informats:

- "S370FIBw.d Informat" on page 316
- "S370FIBUw.d Informat" on page 317

S370FRB*w.d* Informat

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

Category: Numeric

Syntax

S370FRB*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 6

Range: 2–8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

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

Comparisons

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

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

	Real Binary Notation	
Language	**4 Bytes**	**8 Bytes**
C	float	double

See Also

Informats:

- "RBw.d Informat" on page 309

S370FZD*w.d* Informat

Reads zoned decimal data in IBM mainframe format.

Category: Numeric

Syntax

S370FZD*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 1–32

d

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

Default: 0

Range: 0–31

Details

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

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

Comparisons

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

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

Example

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

Data Line *	Result
`----+----1`	
`F1F2C3`	`123`
`F1F2D3`	`-123`

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

See Also

Informats:

- “ZDw.d Informat” on page 351

S370FZDB*w.d* Informat

Reads zoned decimal data in which zeros have been left blank.

Category: Numeric

See: “ZDBw.d Informat: z/OS” in *SAS Companion for z/OS*

Syntax

S370FZBD*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 1–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–31

Details

Use the S370FZDB*w.d* informat on other operating environments to read zoned decimal data from IBM mainframe files.

Example

```
input @1 x s370fzdb8.;
```

Data Line *	Result
----+----1	
40404040F14040C0	1000
4040404040F1F2D3	–123

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

S370FZDL*w.d* Informat

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

Category: Numeric

Syntax

S370FZDL*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 1–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–31

Details

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

Comparisons

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

Example

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

Data Line *	Result
----+----1	
C1F2F3	123
D1F2F3	-123

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

S370FZDS*w.d* Informat

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

Category: Numeric

Syntax

S370FZDS*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 2–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–31

Details

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

Comparisons

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

Example

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

Data Line *	Result
`----+----1`	
`4EF1F2F3`	`123`
`60F1F2F3`	`-123`

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

S370FZDT*w.d* Informat

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

Category: Numeric

Syntax

S370FZDT*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 2–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–31

Details

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

Comparisons

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

Example

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

Data Line *	Result
----+----1	
F1F2F34E	123
F1F2F360	-123

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

S370FZDU*w.d* Informat

Reads unsigned zoned decimal data in IBM mainframe format.

Category: Numeric

Syntax

S370FZDU*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 8

Range: 1–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Default: 0

Range: 0–31

Details

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

Comparisons

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

Example

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

Data Line *	Result
----+----1	
F1F2F3	123

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

SHRSTAMP*w.* Informat

Reads date and time values of SHR records.

Category: Date and Time

Syntax

SHRSTAMP*w*.

Syntax Description

w

specifies the width of the input field.

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

Details

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

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

ccyy

is the two byte representation of the year. The *cc* portion is the one byte representation of a two-digit integer that represents the century. The *yy* portion is the one byte representation of two digits that correspond to the year.

The *cc* portion is the century indicator where 00 indicates 19yy, 01 indicates 20yy, 02 indicates 21yy, and so on. A hexadecimal year value of 0115 is equal to the year 2015.

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

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

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

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

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

th
: is the one byte representation of two digits that correspond to a 100th of a second.

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

Example

```
input begin: $hex16.;
y=input(begin, shrstamp8.);
```

Data Line *	Result
`----+----1----+----2`	
`0110239F12403576`	`1598532035.8`

* The data line is a hexadecimal representation of a packed decimal date and time value that is stored as it would appear in an SHR record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to August. 27, 2010 12:40:36.

SMFSTAMP*w*. Informat

Reads time and date values of SMF records.

Category: Date and Time

Syntax

SMFSTAMP*w*.

Syntax Description

w
: specifies the width of the input field.

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

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

Details

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

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

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

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

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

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

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

Example

```
input begin: $hex16.;
y=input(begin, smfstamp8.);
```

Data Line *	Result
`----+----1----+----2`	
`0058DC0C0108200F`	`1532016635`

* The data line is a hexadecimal representation of a binary time and date value that is stored as it would appear in an SMF record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to July 18, 2008, 4:10:35 PM.

STIMER*w*. Informat

Reads time values and determines whether the values are hours, minutes, or seconds; reads the output of the STIMER system option.

Category: Date and Time

Syntax

STIMER*w*.

Syntax Description

w
specifies the width of the input field.

Details

The STIMER informat reads performance statistics that the STIMER system option writes to the SAS log.

The informat reads time values and determines whether the values are hours, minutes, or seconds based on the presence of decimal points and colons:

- If no colon is present, the value is the number of seconds.
- If a single colon is present, the value before the colon is the number of minutes. The value after the colon is the number of seconds.
- If two colons are present, the sequence of time is hours, minutes, and then seconds.

In all cases, the result is a SAS time value.

The input values for STIMER must be in one of the following forms:

- *ss*
- *ss.ss*
- *mm:ss*
- *mm:ss.ss*
- *hh:mm:ss*
- *hh:mm:ss.ss*

ss
is an integer that represents the number of seconds.

mm
is an integer that represents the number of minutes.

hh
is an integer that represents the number of hours.

TIME*w*. Informat

Reads hours, minutes, and seconds in the form *hh:mm:ss.ss*, where special characters such as the colon (:) or the period (.) are used to separate the hours, minutes, and seconds.

Category: Date and Time

Syntax

TIME*w*.

Syntax Description

w
specifies the width of the input field.
Default: 8

Range: 5–32

Details

The TIME*w*. informat reads SAS time values in the form: *hh:mm:ss*<*.ss*> <AM | PM>:

hh
: is an integer that represents the number of hours.

:
: represents a special character that separates hours, minutes, and seconds.

mm
: is an integer between 00 and 59 that represents minutes.

ss<*.ss*>
: is an integer that represents the number of seconds, and if needed, tenths of a second. Seconds and tenths of a second must always be separated by a period.

AM | PM
: AM indicates time between 12:00 midnight and 11:59 in the morning. PM indicates time between 12:00 noon and 11:59 at night.

Separate *hh*, *mm*, and *ss* with a special character. When the period is used as the special character, the time is interpreted in the order hours, minutes, and seconds. For example, 23.22 is 23 hours and 22 minutes, not 23 minutes and 22 seconds, or 23 seconds and 22 tenths of a second.

If you do not enter a value for seconds, SAS assumes a value of 0.

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

Example

```
input begin time10.;
```

Data Line	Result	Formatted with TIME*w*.
`----+----1`		
`12.56`	`46560`	`12:56:00`
`120:120`	`439200`	`122:00:00`
`1:13 pm`	`47580`	`13:13:00`

See Also

Formats:

- “HHMMw.d Format” on page 103
- “HOURw.d Format” on page 105
- “MMSSw.d Format” on page 117
- “TIMEw.d Format” on page 157

Functions:

- “HOUR Function” in *SAS Functions and CALL Routines: Reference*
- “MINUTE Function” in *SAS Functions and CALL Routines: Reference*
- “SECOND Function” in *SAS Functions and CALL Routines: Reference*
- “TIME Function” in *SAS Functions and CALL Routines: Reference*

TODSTAMP*w*. Informat

Reads an eight-byte time-of-day stamp.

Category: Date and Time

Syntax

TODSTAMP*w*.

Syntax Description

w

specifies the width of the input field.

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

Details

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

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

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

Example

```
input btime: $hex16.;
y=input(btime, todstamp8.);
```

Data Line *	Result
----+----1----+----2	
B591183D5FB80000	1300786905

* The data line is a hexadecimal representation of a binary, 8-byte time-of-day clock value. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to March 21, 2001, 09:41:45.

TRAILSGN*w*. Informat

Reads a trailing plus (+) or minus (–) sign.

Category: Numeric

Syntax

TRAILSGN*w*.

Syntax Description

w

specifies the width of the input field.

Default: 6

Range: 1–32

Details

If the data contains a decimal point, the TRAILSGN informat honors the number of decimal places that are in the input data. If the data contains a comma, the TRAILSGN informat reads the value, ignoring the comma.

Example

```
input x trailsgn8.;
```

Data Line	Result
`----+----1`	
`1`	`1`
`1,000`	`1000`
`1+`	`1`
`1-`	`-1`
`1.2`	`1.2`
`1.2+`	`1.2`
`1.2-`	`-1.2`

TU*w*. Informat

Reads timer units.

Category: Date and Time

Syntax

TU*w*.

Syntax Description

w

specifies the width of the input field.

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

Details

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

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

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

Example

```
input btime tu4.;
```

Data Line *	Result
`----+----1`	
`8FC7A9BC`	`62818.411563`

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

VAXRB*w.d* Informat

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

Category: Numeric

Syntax

VAXRB*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 4

Range: 2–8

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–10

Details

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

Comparisons

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

See Also

Informats:

- "RBw.d Informat" on page 309

VMSZN*w.d* Informat

Reads VMS and MicroFocus COBOL zoned numeric data.

Category: Numeric

Syntax

VMSZN*w.d*

Required Arguments

w

specifies the width of the output field.

Default: 1

Range: 1–32

d

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.

Details

The VMSZN*w.d* informat is similar to the ZD*w.d* informat. Both read a string of ASCII digits, and the last digit is a special character denoting the magnitude of the last digit and the sign of the entire number. The difference between the VMSZN*w.d* informat and the ZD*w.d* informat is in the special character used for the last digit. The following table shows the special characters used by the VMSZN*w.d* informat.

Desired Digit	Special Character	Desired Digit	Special Character
0	0	-0	p
1	1	-1	q
2	2	-2	r

Desired Digit	Special Character	Desired Digit	Special Character
3	3	-3	s
4	4	-4	t
5	5	-5	u
6	6	-6	v
7	7	-7	w
8	8	-8	x
9	9	-9	y

Data formatted using the VMSZN*w.d* informat are ASCII strings.

Example

```
input @1 vmszn4.;
```

Data line	Result
`----+----1`	
`1234`	`1234`
`123t`	`-1234`

See Also

Formats:

- "VMSZNw.d Format" on page 163

Informats:

- "ZDw.d Informat" on page 351

w.d Informat

Reads standard numeric data.

Category: Numeric

Alias: BEST*w.d*, D*w.d*, E*w.d*, F*w.d*

Syntax

w.d

Syntax Description

w

specifies the width of the input field.

Range: 1–32

d

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

Range: 0–31

Details

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

Comparisons

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

Example

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

Data Line	Result
`----+----1----+`	`----+----1----+`
`23        2300`	`23    23`
`  23   2300`	`23    0`
`23       -2300`	`23    -23`
`23.0       23.`	`23    23`
`2.3E1     2.3`	`23    2.3`
`-23     0`	`-23   .`

WEEKU*w*. Informat

Reads a value in the form of a week-number within the year and returns a SAS date value by using the U algorithm.

Category: Date and Time

Syntax

WEEKU*w*.

Syntax Description

w

specifies the width of the input field.

Default: 11

Range: 3–200

Details

The WEEKU*w*. informat reads the week-number value within the year, and then returns a SAS date value by using the U algorithm. If the input does not contain a year expression, then WEEKU*w*. uses the current year as the year expression, which is the default. If the input does not contain a day expression, then WEEKU*w*. uses the first day of the week as the day expression, which is the default.

The U Algorithm calculates the SAS date value using the number-of-week value within the year (Sunday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

The inputs to the WEEKU*w*. informat are the same date for the following example. The current year is 2012.

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	12W01
7-8	yyWwwdd	12W0101
9-10	yyyyWwwdd	2012W0101
11-200	yyyy-Www-dd	2012-W01-01

Comparisons

The WEEKU*w*. informat reads the week-number value as a decimal number in the range 0–53, with Sunday as the first day of the week.

The WEEKV*w*. informat reads the number-of-week value as a decimal number in the range 01–53, with Monday as the first day of the week. Week one of the year is the week that includes both January fourth and the first Thursday of the year. If the first Monday of January is the second, third, or fourth, the preceding days are part of the last week of the preceding year.

The WEEKW*w*. informat reads the week-number value as a decimal number in the range 00–53, with Monday as the first day of week.

Example

The current year is 2012 in the following examples.

Statements	Result
	`----+----1`

`v=input('W01',weeku3.);`
`w=input('03W01',weeku5.);`
`x=input('03W0101',weeku7.);`
`y=input('2003W0101',weeku9.);`
`z=input('2003-W01-01',weeku11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	
	`18993`
	`18993`
	`18993`
	`18993`
	`18993`

See Also

Formats:

- "WEEKUw. Format" on page 169
- "WEEKVw. Format" on page 171
- "WEEKWw. Format" on page 173

Functions:

- "WEEK Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "WEEKVw. Informat" on page 341
- "WEEKWw. Informat" on page 343

WEEKV*w*. Informat

Reads a value in the form a week-number within a year and returns a SAS date value using the V algorithm.

Category: Date and Time

Syntax

WEEKV*w*.

Syntax Description

w

specifies the width of the input field.

Default: 11

Range: 3–200

Details

The WEEKV*w*. informat reads the week-number value within a year. If the input does not contain a year expression, WEEKV*w*. uses the current year as the year expression, which is the default. If the input does not contain a day expression, WEEKV*w*. uses the first day of the week as the day expression, which is the default.

The V algorithm calculates the SAS date value. The number-of-week value is represented as a decimal number in the range 01–53, with a leading zero and maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4th and the first Thursday of the year. If the first Monday of January is the 2nd, 3rd, or 4th, the preceding days are part of the last week of the preceding year. For example, the fifth week of the year would be represented as 06.

The inputs to the WEEKV*w*. informat are the same date for the following example. The current year is 2012.

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	12W01
7-8	yyWwwdd	12W0101
9-10	yyyyWwwdd	2012W0101
11-200	yyyy-Www-dd	2012-W01-01

Comparisons

The WEEKV*w*. informat reads the week-number value as a decimal number in the range 01–53, with Monday as the first day of the week. Week one of the year is the week that includes both January fourth and the first Thursday of the year. If the first Monday of January is the second, third, or fourth, the preceding days are part of the last week of the preceding year.

The WEEKU*w*. informat reads the week-number value as a decimal number in the range 0–53, with Sunday as the first day of the week.

The WEEKW*w*. informat reads the week-number-of-year value as a decimal number in the range 00–53, with Monday as the first day of week.

Example

The current year is 2012 in the following examples.

Statements	Result
	`----+----1`

`v=input('W01',weekv3.);`
`w=input('03W01',weekv5.);`
`x=input('03W0101',weekv7.);`
`y=input('2003W0101',weekv9.);`
`z=input('2003-W01-01',weekv11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	
	`18994`
	`18994`
	`18994`
	`18994`
	`18994`

See Also

Formats:

- "WEEKUw. Format" on page 169
- "WEEKVw. Format" on page 171
- "WEEKWw. Format" on page 173

Functions:

- "WEEK Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "WEEKUw. Informat" on page 340
- "WEEKWw. Informat" on page 343

WEEKW*w*. Informat

Reads a value in the form of a week-number within the year and returns a SAS date value using the W algorithm.

Category: Date and Time

Syntax

WEEKW*w*.

Syntax Description

w

specifies the width of the input field.

Default: 11

Range: 3–200

Details

The WEEKW*w*. informat reads the week-number value within the year. If the input does not contain a year expression, the WEEKW*w*. informat uses the current year as the year expression, which is the default. If the input does not contain a day expression, the WEEKW*w*. informat uses the first day of the week as the day expression, which is the default. Algorithm W calculates the SAS date value using the number of the week within the year (Monday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

The inputs to the WEEKW*w*. informat are the same date for the following example. The current year is 2012.

Widths	Formats	Examples
3-4	Www	w01
5-6	yyWww	12W01
7-8	yyWwwdd	12W0101
9-10	yyyyWwwdd	2012W0101
11-200	yyyy-Www-dd	2012-W01-01

Comparisons

The WEEKW*w*. informat reads the week-number value as a decimal number in the range 00–53, with Monday as the first day of week.

The WEEKU*w*. informat reads the week-number value as a decimal number in the range 00–53, with Sunday as the first day of the week.

The WEEKV*w*. informat reads the week-number value as a decimal number in the range 01–53, with Monday as the first day of the week. Week one of the year is the week that includes both January fourth and the first Thursday of the year. If the first Monday of January is the second, third, or fourth, the preceding days are part of the last week of the preceding year.

Example

The current year is 2012 in the following examples.

Statements	Result
	`----+----1`

Statements	Result

`v=input('W01',weekw3.);`
`w=input('03W01',weekw5.);`
`x=input('03W0101',weekw7.);`
`y=input('2003W0101',weekw9.);`
`z=input('2003-W01-01',weekw11.);`
`put v;`
`put w;`
`put x;`
`put y;`
`put z;`
	
	`18994`
	`18994`
	`18994`
	`18994`
	`18994`

See Also

Formats:

- “WEEKUw. Format” on page 169
- “WEEKVw. Format” on page 171
- “WEEKWw. Format” on page 173

Function:

- “WEEK Function” in *SAS Functions and CALL Routines: Reference*

Informats:

- “WEEKUw. Informat” on page 340
- “WEEKVw. Informat” on page 341

YMDDTTM*w.d* Informat

Reads datetime values in the form *<yy>yy-mm-dd hh:mm:ss.ss*, where special characters such as a hyphen (-), period (.), slash (/), or colon (:) are used to separate the year, month, day, hour, minute, and seconds; the year can be either 2 or 4 digits.

Category: Date and Time

Alignment: right

Syntax

YMDDTTM*w.d*

Syntax Description

w

specifies the width of the output field.

Default: 19

Range: 13–40

d

specifies the number of digits to the right of the decimal point in the seconds value. The digits to the right of the decimal point specify a fraction of a second. This argument is optional.

Default: 0

Range: 0–39

Details

The YMDDTTMw.d format reads SAS datetime values in the form *<yy>yy-mm-dd hh*:*mm*:<*ss*<.*ss*>>:

yy or *yyyy*

specifies a two- or four-digit integer that represents the year.

mm

is an integer between 01 and 12 that represents the month.

dd

is an integer between 01 and 31 that represents the day of the month.

hh

is an integer between 00 and 23 that represents hours.

mm

is an integer between 00 and 59 that represents minutes.

ss.ss

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

requirement	If a fraction of a second is specified, the decimal point can be represented only by a period and is required.

\- or :

represents one of several special characters, such as the slash (/), hyphen (-), colon (:), or a blank character that can be used to separate date and time components. Special characters can be used as separators between any date or time component and between the date and the time.

Comparisons

The YMDDTTM*w.d* informat reads datetime values with required separators in the form *<yy>yy-mm-dd/hh:mm:ss.ss*.

The MDYAMPM*w.d* in format reads datetime values with optional separators in the form *mm-dd-yy<yy> hh:mm:ss.ss* AM | PM, and requires a space between the date and the time.

The DATETIME*w.d* informat reads datetime values with optional separators in the form *dd-mmm-yy<yy> hh:mm:ss.ss* AM|PM, and the date and time can be separated by a special character.

Example

```
input @1 dt ymddttm24.;
```

Data Line	Result
`2012-03-16 11:23:07.4`	`1647516187.4`
`2012 03 16 11 23 07.4`	`1647516187.4`
`12.3.16/11:23`	`1647516180`

See Also

Informats:

- "DATETIMEw. Informat" on page 268
- "MDYAMPMw.d Informat" on page 291

YYMMDD*w*. Informat

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

Category: Date and Time

Syntax

YYMMDD*w*.

Syntax Description

w

specifies the width of the input field.

Default: 6

Range: 6–32

Details

SAS read date values in one of the following forms:

- *yymmdd*
- *yyyymmdd*

yy or *yyyy*

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

mm

is an integer between 01 and 12 that represents the month of the year.

dd

is an integer between 01 and 31 that represents the day of the month.

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

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

Example

```
input calendar_date yymmdd10.;
```

Data Line	Result
----+----1	
120316	19068
12/03/16	19068
12 03 16	19068
2012-03-16	19068

See Also

Formats:

- "DATEw. Format" on page 73
- "DDMMYYw. Format" on page 78
- "MMDDYYw. Format" on page 113
- "YYMMDDw. Format" on page 181

Functions:

- "DAY Function" in *SAS Functions and CALL Routines: Reference*
- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267
- "DDMMYYw. Informat" on page 270
- "MMDDYYw. Informat" on page 292

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

YYMMN*w*. Informat

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

Category: Date and Time

Syntax

YYMMN*w*.

Syntax Description

w
specifies the width of the input field.

Default: 4

Range: 4–6

Details

SAS reads date values in one of the following forms:

- *yyyymm*
- *yymm*

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

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

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

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

Example

```
input date1 yymmn6.;
```

Data Line	Result
----+----1	
201208	19206

See Also

Formats:

- "DATEw. Format" on page 73
- "DDMMYYw. Format" on page 78
- "YYMMDDw. Format" on page 181
- "YYMMw. Format" on page 180
- "YYMONw. Format" on page 186

Functions:

- "DAY Function" in *SAS Functions and CALL Routines: Reference*
- "MONTH Function" in *SAS Functions and CALL Routines: Reference*
- "MDY Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*

Informats:

- "DATEw. Informat" on page 267
- "DDMMYYw. Informat" on page 270
- "MMDDYYw. Informat" on page 292
- "YYMMDDw. Informat" on page 347

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

YYQ*w*. Informat

Reads quarters of the year in the form *yy*Q*q* or *yyyy*Q*q*.

Category: Date and Time

Syntax

YYQ*w*.

Syntax Description

w

specifies the width of the input field.

Default: 6 (For SAS version 6, the default is 4.)

Range: 4–32 (For SAS version 6, the range is 4–6.)

Details

SAS reads data in one of the following forms:

- *yy*Q*q*
- *yyyy*Q*q*

yy or *yyyy*

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

q

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

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

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

Example

```
input quarter yyq9.;
```

Data Line	Result
`----+----1----+`	
12Q2	19084
12Q02	19084
2012Q02	19084

See Also

Functions:

- "QTR Function" in *SAS Functions and CALL Routines: Reference*
- "YEAR Function" in *SAS Functions and CALL Routines: Reference*
- "YYQ Function" in *SAS Functions and CALL Routines: Reference*

System Options:

- "YEARCUTOFF= System Option" in *SAS System Options: Reference*

ZD*w.d* Informat

Reads zoned decimal data.

Category: Numeric

See: "ZDw.d Informat: UNIX" in *SAS Companion for UNIX Environments*
"ZDw.d Informat: Windows" in *SAS Companion for Windows*
"ZDw.d Format: z/OS" in *SAS Companion for z/OS*

Syntax

ZD*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 1–31

Details

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

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

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

Comparisons

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

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

Example

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

Data Line *	Result
----+----1	
F0F1F2C8	128

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

See Also

Informats

- "w.d Informat" on page 338
- "ZDVw.d Informat" on page 353

ZDB*w.d* Informat

Reads zoned decimal data in which zeros have been left blank.

Category: Numeric

See: "ZDBw.d Informat: z/OS" in *SAS Companion for z/OS*

Syntax

ZDB*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 0–31

Details

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

Example

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

Data Line *	Result
----+----1	
F140C2	102

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

ZDV*w.d* Informat

Reads and validates zoned decimal data.

Category: Numeric

Syntax

ZDV*w.d*

Syntax Description

w

specifies the width of the input field.

Default: 1

Range: 1–32

d

specifies the power of 10 by which to divide the value. This argument is optional.

Range: 1–31

Details

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

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

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

Comparisons

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

Example

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

Data Line *	Result
----+----1	
F0F1F2C8	128

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

See Also

Informats:

- "w.d Informat" on page 338

- "ZDw.d Informat" on page 351

Index

H

I

J

L