# SAS® 9.4 Formats and Informats: Reference

SAS® Documentation
January 19, 2024

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® 9.4 Formats and Informats: Reference*. Cary, NC: SAS Institute Inc.

**SAS® 9.4 Formats and Informats: Reference**



SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

February 2024



9.4-P10:leforinforref

# Contents

# 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 (=). The syntax for arguments has multiple forms in order to demonstrate the syntax of multiple arguments, with and without punctuation.

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 these examples of SAS syntax, the keywords are bold:

  **CHAR** (*string, position*)

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

  **ALTER** (*alter-password*)

  **BEST** *w*.

**REMOVE** <*data-set-name*>

In this 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**

Some procedure statements have multiple keywords throughout the statement syntax:

**CREATE** <UNIQUE> **INDEX** *index-name* **ON** *table-name* (*column-1* <, *column-2*, …>)

*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 in angle brackets ( < > ).

In this 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 this 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 this example, *string* and *substring* are required arguments, whereas *modifiers* and *startpos* are optional.

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

*argument(s)*
specifies that one argument is required and that multiple arguments are allowed. Separate arguments with a space. Punctuation, such as a comma ( , ) is not required between arguments.

The MISSING statement is an example of this form of multiple arguments:

**MISSING** *character(s)*;

<LITERAL_ARGUMENT> *argument-1* <<LITERAL_ARGUMENT> *argument-2* ... >
specifies that one argument is required and that a literal argument can be associated with the argument. You can specify multiple literals and argument pairs. No punctuation is required between the literal and argument pairs. The ellipsis (...) indicates that additional literals and arguments are allowed.

The BY statement is an example of this argument:

**BY** <DESCENDING> *variable-1* <<DESCENDING> *variable-2* …>;

*argument-1 <options> <argument-2 <options> ...>*
specifies that one argument is required and that one or more options can be associated with the argument. You can specify multiple arguments and associated options. No punctuation is required between the argument and the option. The ellipsis (...) indicates that additional arguments with an associated option are allowed.

The FORMAT procedure PICTURE statement is an example of this form of multiple arguments:

**PICTURE** name <(*format-options*)>
<*value-range-set-1* <(*picture-1-options*)>
<*value-range-set-2* <(*picture-2-options*)> …>>;

*argument-1=value-1 <argument-2=value-2 ...>*
specifies that the argument must be assigned a value and that you can specify multiple arguments. The ellipsis (...) indicates that additional arguments are allowed. No punctuation is required between arguments.

The LABEL statement is an example of this form of multiple arguments:

**LABEL** *variable-1=label-1 <variable-2=label-2* …>;

*argument-1 <, argument-2, ...>*
specifies that one argument is required and that you can specify multiple arguments that are separated by a comma or other punctuation. The ellipsis (...) indicates a continuation of the arguments, separated by a comma. Both forms are used in the SAS documentation.

Here are examples of this form of multiple arguments:

**AUTHPROVIDERDOMAIN** (*provider-1:domain-1* <, *provider-2:domain-2*, …>

**INTO** :*macro-variable-specification-1* <, :*macro-variable-specification-2*, …>

**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 this example, the keyword ERROR is written in uppercase bold:

**ERROR** <*message*>;

UPPERCASE
identifies arguments that are literals.

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

**CMPMODEL**=BOTH | CATALOG | XML |

*italic*

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

- nonliteral arguments. In this example of the LINK statement, the argument *label* is a user-supplied value and therefore appears in italic:

  **LINK** *label*;

- nonliteral values that are assigned to an argument.

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

  **FORMAT** *variable(s)* <*format* > <DEFAULT = *default-format*>;

# 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 this example of the MAPS system option, the equal sign sets the value of MAPS:

**MAPS**=*location-of-maps*

< >

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

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

**CAT** (*item-1* <, *item-2*, …>)

|

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

**CMPMODEL**=BOTH | CATALOG | XML

...

an ellipsis indicates that the argument can be repeated. If an argument and the ellipsis are enclosed in angle brackets, then the argument is optional. The repeated argument must contain punctuation if it appears before or after the argument.

In this example of the CAT function, multiple *item* arguments are allowed, and they must be separated by a comma:

**CAT** (*item-1* <, *item-2*, …>)

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

In this 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 this 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 typically have a choice of using a SAS statement (LIBNAME or FILENAME) or the operating environment's control language to make the reference. 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* enclosed in quotation marks:

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

# What's New in SAS 9.4 Formats and Informats

## Overview

SAS Viya 3.4 adds the $UUID format and informat.

New formats write date, time, and datetime values based on user local time.

Some format values might differ slightly when the DECIMALCONV= system option is set to STDIEEE.

## New SAS Formats

The following formats are new:

$UUID
: converts character data to Universally Unique Identifier (UUID) format.

B8601DX*w*.
: adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation *yyyymmdd*T*hhmmss*+*hhmm*.

B8601LX*w*.
: writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation *yyyymmdd*T*hhmmss*+–-*hhmm*.

B8601TX*w*.
: adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 basic time notation *hhmmss*+|–*hhmm*.

E8601DX*w*.
: adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone extended notation *yyyy-mm-dd*T*hh:mm:ss*+*hh:mm*.

E8601LX*w*.
: writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation *yyyy-mm-dd*T*hh:mm:ss*+|–*hh:mm*.

E8601TX*w*.
: adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 extended time notation *hh:mm:ss*+|–*hh:mm*.

ODDSR*w*.
: writes odds ratios.

# Enhancements to Formats

In 9.4m7, the F format is added to the documentation. This format is an alias for the W format.

# Format Output Differences Due to the DECIMALCONV= System Option

When the DECIMALCONV= system option is set to STDIEEE, SAS converts and formats decimal values using the IEEE Standard for Floating-Point Arithmetic 754–2008. This standard improves the accuracy of floating-point numbers. The output written for the following formats might differ slightly from previous releases:

| | | |
|---|---|---|
| BESTD*w.p* | E8601TZ*w.d* | S370FPDU*w.d* |
| B8601DT*w.d* | HHMM*w.d* | S370FZD*w.d* |
| B8601TM*w.d* | HOUR*w.d* | S370FZDL*w.d* |
| COMMA*w.d* | MDYAMPM*w.d* | S370FZDS*w.d* |
| D*w.d* | MMSS*w.d* | S370FZDT*w.d* |
| DATEAMPM*w.d* | NEGPAREN*w.d* | S370FZDU*w.d* |
| DATETIME*w.d* | NUMX*w.d* | TIME*w.d* |
| DOLLAR*w.d* | PD*w.d* | TIMEAMPM |
| DOLLARX*w.d* | PERCENT*w.d* | TOD |
| E*w.* | PERCENTN*w.d* | *w.d* |
| E8601DT*w.d* | PVALUE*w.d* | ZD*w.d* |
| E8601TM*w.d* | S370FPD*w.d* | |

For more information, see “DECIMALCONV=” in *SAS System Options: Reference*.

# New Informats

SAS Viya 3.4 has added the $UUID*w*. informat.

$UUID*w*.
converts hexadecimal data to a 16-byte binary variable Universally Unique Identifier (UUID).

# Enhancements to Existing Informats

SAS 9.4M2 has added new aliases for some time-zone informats:

| New Alias | Existing Informat |
|---|---|
| B8601DXw. on page 602 | B8601DZw.d on page 603 |
| B8601LXw. on page 605 | B8601DTw.d on page 601 |
| B8601TXw. on page 607 | B8601TZw.d on page 607 |
| E8601DXw. on page 627 | E8601DZw.d on page 627 |
| E8601LXw. on page 629 | E8601DTw.d on page 625 |
| E8601TXw. on page 633 | E8601TZw.d on page 633 |

# PART 1

# SAS Formats

# 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 type of data: 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. 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*>

**$**
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 "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 contain a period (.) as a part of the name. If you omit the *w* and *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 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 any commas.

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 this example, the result is x=**:

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

If you use an incompatible format, such as 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 is displayed in the SAS log.

When the value of *d* is greater than 15, 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 specify formats in these ways:

- in a PUT statement
- with the PUT, PUTC, or PUTN functions
- with the %SYSFUNC or %QSYSFUNC macro functions
- 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" in *SAS DATA Step Statements: Reference*.

# PUT Function

The PUT function converts a numeric variable, a character variable, or a constant by using any valid format, and then returns the resulting character value. For example, this statement converts the value of a numeric variable to 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" 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 function outside a DATA step. For example, this 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, %QSYSFUNC" 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" in *SAS DATA Step 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 that use a FORMAT statement behave like formats that are used with a colon (:) modifier in a subsequent PUT statement. For more information about using the colon format modifier, see "PUT: List" in *SAS DATA Step Statements: Reference*.

## ATTRIB Statement

The ATTRIB statement can also associate a format, as well as other attributes, with one or more variables. In this 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" in *SAS DATA Step 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 *Base SAS Procedures Guide*.

# User-Defined Formats

In addition to the formats that are supplied with Base SAS software, you can use PROC FORMAT to create your own formats. PROC FORMAT enables you to create your own formats for both character and numeric variables.

**Note:** PROC FORMAT is valid on the SAS client and the CAS server.

For more information, see "FORMAT Procedure" in *Base SAS Procedures Guide* and *SAS Cloud Analytic Services: User-Defined Formats*.

User-defined format names cannot end in a number. If you specify a user-defined format whose name ends in a number, SAS returns an error. You can re–create the format with a valid name that does not end in a number, or you can use PROC CATALOG to rename the format. Here is an example that uses PROC CATALOG to change the name of a user-defined format from MYFMT22 to MYFMT.

```
proc catalog cat=work.formats;
change myfmt22=myfmt / entrytype = format;
quit;
```

**Note:** A SAS user or a third party's software might create a format with a name that ends in a number. Follow the guidelines for naming SAS elements at "Words and Names" in *SAS Programmer's Guide: Essentials* when you create user-defined formats.

When you execute a SAS program on the SAS client that uses user-defined formats, you can make these formats available in two ways:

- Create permanent, not temporary, formats with PROC FORMAT.
- 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 "FORMAT Procedure" in *Base SAS Procedures Guide*.

Your CAS administrator can configure the server so that format libraries are added and promoted from libraries that are persisted as SASHDAT files in a caslib's data source. The format libraries that you have created are not automatically loaded when you start a CAS session, even if you have persisted them.

When you execute a SAS program in a CAS session that uses user-defined formats, you can make these formats available in three ways:

- Use the addFmtLib action to add a format library that has been created with PROC FORMAT.
- Use the SAVEFMTLIB option of the CAS statement to save a format library to a CAS table or a SASHDAT. The table and the file include the formats in a format library.

- Use the LOADFORMATS option of the CAS statement to load user-defined formats from a format item-store file that was created with PROC FMTC2ITM.

For more information, see addFmtLib Action and CAS Statement.

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, these system options produce these results:

| System Option | Result |
|---|---|
| FMTERR | SAS produces an error that causes the current DATA or PROC step to stop processing. |
| NOFMTERR | SAS continues processing and substitutes a default format, usually the BEST*w*. or $*w*. format. SAS also writes a note to the log indicating that the specified format was not found or could not be loaded. |

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

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

# Definitions for Dates, Times, and Intervals

## Definitions

SAS date value
: is a value that represents the number of days between January 1, 1960, and a specified date. SAS can perform calculations on dates that range from A.D. November 1582 to A.D. 4000. Dates before January 1, 1960, are negative numbers; dates after January 1, 1960, are positive numbers.

- SAS date values account for all leap year days, including the leap year day in the year 2000.

**CAUTION**
**Using century dates greater than 4000 might result in incorrect dates.** SAS does not consider century years that are divisible by 4000 to be leap years. Computations on dates that use a century date greater than or equal to 4000 might be off by days, depending on the computation. SAS does not consider the years 4000, 8000, 12000, 16000, and 20000 to be leap years.

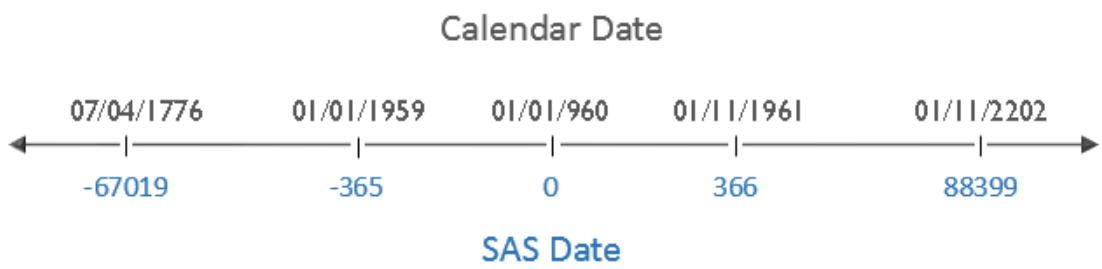
- SAS date values can reliably tell you what day of the week a particular day fell on as far back as September 1752. That was when the calendar was adjusted by dropping several days. SAS day-of-the-week and length-of-time calculations are accurate in the future to A.D. 19,900.
- Various SAS language elements handle SAS date values: functions, formats, and informats.

SAS time value
: is a value that represents the number of seconds since midnight of the current day. SAS time values are between 0 and 86400.

SAS datetime value
: is a value that represents the number of seconds between January 1, 1960, and an hour/minute/second within a specified date.

The following figure shows some dates written in calendar form and as SAS date values.

***Figure 1.1*** *How SAS Converts Calendar Dates to SAS Date Values*




# Julian Date Formats and Astronomical Dates

SAS uses SAS date values, which are ordinal numbers, to calculate dates. SAS date values represent the number of days between January 1, 1960, and a specified date. All SAS formats, informats, and functions use SAS dates. You have to use an informat to convert a Julian date to a SAS date before SAS can use it to perform calculations.

The following SAS language elements do not convert SAS dates to Julian dates. These language elements apply a Julian date format to a SAS date.

| Format | Informat | Function |
|---|---|---|
| JULDAY | JULIAN | DATEJUL |
| JULIAN | PDJULG | JULDATE |
| PDJULG | PDJULI | JULDATE7 |
| PDJULI | | |

SAS can perform calculations on raw SAS date values and on formatted SAS date values. This includes performing calculations on Julian-formatted date values.

SAS uses these definitions of Julian dates and Julian formats:

Julian date
: is the number of continuous days since January 1, 4713 BC, which is also known as an astronomical date.

Julian format
: is the representation of an ordinal SAS date in the form of a calendar day, YYDDD, or YYDD.

SAS uses the Julian format (ordinal date) definition of dates. Julian-related language elements in SAS do not convert SAS dates internally to Julian astronomical dates. These Julian-related language elements make a SAS date look like an ordinal date with the form YYDDD or YYYYDDD. For example, January, 23, 2018 is 18023 when you apply a Julian format in SAS.

You must define the values as SAS dates before using them in calculations. The only way you can convert a SAS date to an astronomical date is to add 2,436,934.5 to the SAS date value. This conversion enables SAS to use the values to perform calculations. Otherwise, SAS treats the values as regular integer numeric values, and you might get unexpected results.

The following example performs these tasks:

- Creates a data set that contains SAS dates.
- Converts the dates to the MMDDYY10 format and the Julian format.
- Performs calculations on the two sets of dates, even though they have different formats.

```
data dates;  /* 1 */
   input sas_date;
datalines;
21519
21522
21528
21535
21545
21555
21565
;
proc print data = dates;
run;

data dates2;  /* 2 */
   set dates;
   formatted_sas_date = sas_date;
   julian_formatted_SAS_date = sas_date;
   format formatted_sas_date mmddyy10. julian_formatted_SAS_date
julian.;  /* 3 */
run;
proc print data=dates2;
run;

data dates3;  /* 4 */
   set dates2;
   datediff=sas_date - lag(julian_formatted_SAS_date);  /* 5 */
```

```
run;
proc print data =dates3;
run;
```

The following items correspond to the lines that are numbered in the preceding DATA step.

1. Create the data set of SAS dates.
2. Convert the dates to the MMDDYY10 format and the Julian format.
3. The FORMAT statement creates the two formats for the dates.
4. Perform calculations on the formatted dates.
5. The LAG function compares the SAS date values in the previous row with the value in the current row and returns the difference.

***Figure 1.2*** *Converting SAS Dates and Using the Results in Calculations*

| Obs | sas_date |
|---|---|
| 1 | 21519 |
| 2 | 21522 |
| 3 | 21528 |
| 4 | 21535 |
| 5 | 21545 |
| 6 | 21555 |
| 7 | 21565 |

**The SAS System**

| Obs | sas_date | formatted_sas_date | julian_formatted_SAS_date |
|---|---|---|---|
| 1 | 21519 | 12/01/2018 | 18335 |
| 2 | 21522 | 12/04/2018 | 18338 |
| 3 | 21528 | 12/10/2018 | 18344 |
| 4 | 21535 | 12/17/2018 | 18351 |
| 5 | 21545 | 12/27/2018 | 18361 |
| 6 | 21555 | 01/06/2019 | 19006 |
| 7 | 21565 | 01/16/2019 | 19016 |

**The SAS System**

| Obs | sas_date | formatted_sas_date | julian_formatted_SAS_date | datediff |
|---|---|---|---|---|
| 1 | 21519 | 12/01/2018 | 18335 | . |
| 2 | 21522 | 12/04/2018 | 18338 | 3 |
| 3 | 21528 | 12/10/2018 | 18344 | 6 |
| 4 | 21535 | 12/17/2018 | 18351 | 7 |
| 5 | 21545 | 12/27/2018 | 18361 | 10 |
| 6 | 21555 | 01/06/2019 | 19006 | 10 |
| 7 | 21565 | 01/16/2019 | 19016 | 10 |

# Two-Digit and Four-Digit Years

SAS software can read two-digit or four-digit year values. If SAS encounters a two-digit year, the YEARCUTOFF= option can be used to specify which century within a 100-year span the two-digit year should be attributed to. For example, YEARCUTOFF=1950 means that two-digit years 50 through 99 correspond to 1950 through 1999. Two-digit years 00 through 49 correspond to 2000 through 2049. Note that while the default value of the YEARCUTOFF= option in SAS is 1926, you can adjust the YEARCUTOFF= value in a DATA step to accommodate the range of date values that you are working with. To correctly handle two-digit years that represent dates between 2000 and 2099, you should specify an appropriate YEARCUTOFF= value between 1901 and 2000. For more information, see the "YEARCUTOFF=" in *SAS System Options: Reference*.

# Five-Digit Years

Although some formats specify a width large enough to accommodate formatting a five-digit year such as DATETIME20., SAS documentation does not display five-digit years.

# The Year 2000

## Using the YEARCUTOFF= System Option

SAS software treats the year 2000 like any other leap year. If you use two-digit year numbers for dates, you probably need to adjust the default setting for the YEARCUTOFF= option to work with date ranges for your data or switch to four-digit years. The following program changes the YEARCUTOFF= value to 1950. This change means that all two-digit dates are now assumed to fall in the 100-year span from 1950 to 2049.

```
options yearcutoff=1950;
data _null_;
   a='26oct02'd;
   put 'SAS date='a;
   put 'formatted date='a date9.;
run;
```

The PUT statement writes the following lines to the SAS log:

```
SAS date=15639
formated date=26OCT2002
```

**Note:** Whenever possible, specify a year using all four digits. Most SAS date and time language elements support four-digit year values.

## Example: How YEARCUTOFF= Affects Two- and Four-Digit Years

The following example shows what happens with data that contains both two- and four-digit years. By default, the YEARCUTOFF= option is set to 1926.

```
options nodate;


data schedule;
   input @1 jobid $ @6 projdate mmddyy10.;
   datalines;
A100 01/15/25
A110 03/15/2025
A200 01/30/96
B100 02/05/12
B200 06/15/2012
;

proc print data=schedule;
   format projdate mmddyy10.;
run;
```

The output from the PROC PRINT statement looks like this.

***Output 1.1*** *Output Showing Four-Digit Years That Result from Setting YEARCUTOFF= to 1926*

**The SAS System**

| Obs | jobid | projdate |
|---|---|---|
| 1 | A100 | 01/15/2025 |
| 2 | A110 | 03/15/2025 |
| 3 | A200 | 01/30/1996 |
| 4 | B100 | 02/05/2012 |
| 5 | B200 | 06/15/2012 |

Here are some facts to note in this example:

- In the data lines in the DATA step, the first record contains a two-digit year of 25, and the second record contains a four-digit year of 2025. The century for the first record defaults to the 2000s because 2025 is in the range of 1926–2025. The four-digit year in the second record is unaffected by the YEARCUTOFF= option.
- In the third record, the century defaults to the 1900s because the year 1996 is in the range of 1926–2025.

- The output from the fourth and fifth records shows results that are similar to the first and second records. The fourth record specifies a two-digit year of 12, and the fifth record specifies a four-digit year of 2012. The century in the fourth record defaults to the 2000s because 2012 is in the range of 1926–2025. The four-digit year in the fifth record is unaffected by the YEARCUTOFF= option.

As you can see, specifying a two-digit year might or might not result in the intended century prefix. The optimal value of the YEARCUTOFF= option depends on the range of the dates that you are processing.

In releases SAS 6.06 through SAS 6.12, the default value for the YEARCUTOFF= system option is 1900. Starting with SAS 7, the default value is 1920, and starting with SAS 9.4, the default value is 1926.

For more information about how SAS handles dates, see “Definitions for Dates, Times, and Intervals”.

## Practices That Help Ensure Date Integrity

The following practices help ensure that your date values are correct during all the conversions that occur during processing:

- Store dates as SAS date values, not as simple numeric or character values.
- Use the YEARCUTOFF= system option when converting two-digit dates to SAS date values.
- Examine sets of raw data that come into your SAS process to make sure that any dates containing two-digit years are correctly interpreted by the YEARCUTOFF= system option. Be aware of these situations:
  - Two-digit years that are distributed over more than a 100-year period. For dates that cover more than a 100-year span, you must use either four-digit years in the data or conditional logic in a DATA step to interpret them correctly.
  - Two-digit years that need an adjustment to the default YEARCUTOFF= range. For example, if the default value for YEARCUTOFF= in your operating environment is 1926 and you have a two-digit date in your data that represents 1925, you have to adjust your YEARCUTOFF= value downward by a year in the SAS program that processes this value.
- Ensure that output SAS data sets represent dates as SAS date values.
- Check your SAS programs to ensure that formats and informats that use two-digit years, such as DATE7., MMDDYY6., or MMDDYY8., are reading and writing data correctly.

**Note:** The YEARCUTOFF= option has no effect on dates that are already stored as SAS date values.

# Working with SAS Dates and Times

## Informats and Formats

SAS converts date, time, and datetime values back and forth between calendar dates and clock times with SAS language elements called formats and informats.

- Formats present a value such as a time or date value that is recognized by SAS as a calendar date or clock time in a variety of lengths and notations.
- Informats read notations or a value such as a clock time or a calendar date that might be in a variety of lengths, and then convert the data to a SAS date, time, or datetime value.

SAS can read date and time values that are delimited by the following characters:

! # $ % & ( ) * + - . / : ; < = > ? [ \ ] ^ _ { | } ~

The blank character can also be used.

Only one delimiter can be used for a date. Otherwise, an error message is written to the SAS log. For example, 01/Jan/2007 uses a single delimiter and can be read by SAS. In the case of 01-Jan/2007, two different delimiters separate the parts of the date, which results in an error message.

## Date and Time Tools by Task

The following table correlates tasks with various SAS language elements that are available for working with time and date data.

***Table 1.1*** *Tasks with Dates and Times, Part 1*

| Task | Type of Language Element | Language Element | Input | Result |
|---|---|---|---|---|
| Write SAS date values in recognizable forms | Date formats | DATE. | 19434 | 17MAR13 |
| | | DATE9. | 19434 | 17MAR2013 |
| | | DAY. | 19434 | 17 |
| | | DDMMYY. | 19434 | 17/03/13 |
| | | DDMMYY10. | 19434 | 17/03/2013 |
| | | DDMMYYB. | 19434 | 17 03 13 |
| | | DDMMYYB10. | 19434 | 17 03 2013 |

| Task | Type of Language Element | Language Element | Input | Result |
|---|---|---|---|---|
| | | DDMMYYC. | 19434 | 17:03:13 |
| | | DDMMYYC10. | 19434 | 17:03:2013 |
| | | DDMMYYD. | 19434 | 17-03-13 |
| | | DDMMYYD10. | 19434 | 17-03-2013 |
| | | DDMMYYN. | 19434 | 17032013 |
| | | DDMMYYN6. | 19434 | 170313 |
| | | DDMMYYP. | 19434 | 17.03.13 |
| | | DDMMYYP10. | 19434 | 17.03.2013 |
| | | DDMMYYS. | 19434 | 17/03/13 |
| | | DDMMYYS10. | 19434 | 17/03/2013 |
| | | DOWNAME. | 19434 | Sunday |
| | | JULDAY. [1] | 19434 | 76 |
| | | JULIAN. [1] | 19434 | 13076 |
| | | MMDDYY. | 19434 | 03/17/13 |
| | | MMDDYY10. | 19434 | 03/17/2013 |
| | | MMDDYYB. | 19434 | 03 17 13 |
| | | MMDDYYB10. | 19434 | 03 17 2013 |
| | | MMDDYYC. | 19434 | 03:17:13 |
| | | MMDDYYC10. | 19434 | 03:17:2013 |
| | | MMDDYYD. | 19434 | 03-17-13 |
| Write SAS date values in recognizable forms | Date formats | MMDDYYD10. | 19434 | 03-17-2013 |
| | | MMDDYYN. | 19434 | 03172013 |
| | | MMDDYYN8. | 19434 | 03172013 |
| | | MMDDYYP. | 19434 | 03.17.13 |
| | | MMDDYYP10. | 19434 | 03.17.2013 |
| | | MMDDYYS. | 19434 | 03/17/13 |

| Task | Type of Language Element | Language Element | Input | Result |
|---|---|---|---|---|
| | | MMDDYYS10. | 19434 | 03/17/2013 |
| | | MMYY. | 19434 | 03M2013 |
| | | MMYYC. | 19434 | 03:2013 |
| | | MMYYD. | 19434 | 03-2013 |
| | | MMYYN. | 19434 | 032013 |
| | | MMYYP. | 19434 | 03.2013 |
| | | MMYYS. | 19434 | 03/2013 |
| | | MONNAME. | 19434 | March |
| | | MONTH. | 19434 | 3 |
| | | MONYY. | 19434 | MAR13 |
| | | PDJULG. [1] | 19434 | 2013076F |
| | | PDJULI. [1] | 19434 | 0100076F |
| | | WEEKDATE. | 19434 | Sunday, March 17, 2013 |
| | | WEEKDAY. | 19434 | 1 |
| | | WORDDATE. | 19434 | March 17, 2013 |
| | | WORDDATX. | 19434 | 17 March 2013 |
| | Quarter formats | QTR. | 19434 | 1 |
| | | QTRR. | 19434 | I |
| | Time formats | TIME. | 19434 | 5:23:54 |
| | | TIMEAMPM. | 19434 | 5:23:54 AM |
| | | TOD. | 19434 | 05:23:54 |
| Write SAS date values in recognizable forms | Year formats | YEAR. | 19434 | 2013 |
| | | YYMM. | 19434 | 2013M03 |
| | | YYMMC. | 19434 | 2013:03 |
| | | YYMMD. | 19434 | 2013-03 |
| | | YYMMP. | 19434 | 2013.03 |

| Task | Type of Language Element | Language Element | Input | Result |
|---|---|---|---|---|
| | | YYMMS. | 19434 | 2013/03 |
| | | YYMMN. | 19434 | 201303 |
| | | YYMMDD. | 19434 | 13-03-17 |
| | | YYMON. | 19434 | 2013MAR |
| | Year/Quarter formats | YYQ. | 19434 | 2013Q1 |
| | | YYQC. | 19434 | 2013:1 |
| | | YYQD. | 19434 | 2013-1 |
| | | YYQP. | 19434 | 2013.1 |
| | | YYQS. | 19434 | 2013/1 |
| | | YYQN. | 19434 | 20131 |
| | | YYQR. | 19434 | 2013QI |
| | | YYQRC. | 19434 | 2013:I |
| | | YYQRD. | 19434 | 2013-I |
| | | YYQRP. | 19434 | 2013.I |
| | | YYQRS. | 19434 | 2013/I |
| | | YYQRN. | 19434 | 2013I |

1 In SAS, a Julian date is a date in the form YYNNN or YYYYNNN, where YY is a two-digit year, YYYY is a four-digit year, and NNN is the ordinal offset from January 1 of the year YY or YYYY. SAS processes Julian dates only for valid SAS dates.

***Table 1.2*** *Tasks with Dates and Times, Part 2*

| Task | Type of Language Element | Language Element | Input | Result |
|---|---|---|---|---|
| Date Tasks | | | | |
| Read calendar dates as SAS date<br>Note: YEARCUTOFF=1926 | Date informats | DATE. | 17MAR13 | 19434 |
| | | DATE9. | 17MAR2013 | 19434 |
| | | DDMMYY. | 170313 | 19434 |

| Task | Type of Language Element | Language Element | Input | Result |
|---|---|---|---|---|
| | | DDMMYY8. | 17032013 | 19434 |
| | | JULIAN. [1] | 13076 | 19434 |
| | | JULIAN7. [1] | 2013077 | 19434 |
| | | MMDDYY. | 031713 | 19434 |
| | | MMDDYY8. | 03172013 | 19434 |
| | | MONYY. | MAR13 | 19418 |
| | | YYMMDD. | 130317 | 19434 |
| | | YYMMDD8. | 20130317 | 19434 |
| | | YYQ. | 13q1 | 19359 |
| | | DATETIME | 17MAR2013 00:00:00 | 1679097600 |
| | | TIME | 14:45:32 | 53132 |
| Return today's date as a SAS date value | Date functions | DATE() or TODAY() (equivalent) | ( ) | The SAS date value for today |
| Extract calendar dates from SAS | Date functions | DAY | 19434 | 17 |
| | | HOUR | 19434 | 5 |
| | | JULDATE [1] | 19434 | 13076 |
| | | JULDATE7 [1] | 19434 | 2013076 |
| | | MINUTE | 19434 | 23 |
| | | MONTH | 19434 | 3 |
| | | QTR | 19434 | 1 |
| | | SECOND | 19434 | 54 |
| | | WEEKDAY | 19434 | 1 |
| | | YEAR | 19434 | 2013 |
| Write a date as a constant in an expression | SAS date constant | '*ddmmmyy*'d or '*ddmmmyyyy*'d | '17mar13'd '17mar2013'd | 19434 |

| Task | Type of Language Element | Language Element | Input | Result |
|---|---|---|---|---|
| Write today's date as a string | SYSDATE automatic macro variable | SYSDATE | &SYSDATE | The date at the time of SAS initialization in the form DDMMMYY |
| | SYSDATE9 | SYSDATE9 | &SYSDATE9 | The date at time of SAS initialization, in the form DDMMMYYYY |
| Time Tasks | | | | |
| Write SAS time values as time values | Time formats | HHMM. | 19434 | 5:24 |
| | | HOUR. | 19434 | 5 |
| | | MMSS. | 19434 | 323 |
| | | TIME. | 19434 | 5:23:54 |
| | | TIMEAMPM. | 19434 | 5:23:54 AM |
| | | TOD. | 19434 | 05:23:54 |
| Read time values as SAS time values | Time informats | TIME. | 05:23:54 | 19434 |
| Write the current time as a string | SYSTIME automatic macro variable | SYSTIME | &SYSTIME | The time at the moment of execution, in the form HH:MM |
| Return the current time of day as a SAS time value | Time functions | TIME( ) | ( ) | The SAS time value at the moment of execution, in the form NNNNN.NNN |
| Return the time part of a SAS datetime value | Time functions | TIMEPART | 17mar2013 05:11:43 | 5:11:43 |
| Datetime Tasks | | | | |
| Write SAS datetime values as datetime values | Datetime formats | DATEAMPM | 1679097600 | 17MAR13:12:00:00 AM |

| Task | Type of Language Element | Language Element | Input | Result |
|---|---|---|---|---|
| | | DATETIME | 1679097600 | 17MAR13:00:00:00 |
| Read datetime values as SAS datetime values | Datetime informats | DATETIME | 17MAR13:00:00:00 | 1679097600 |
| Return the current date and time of day as a SAS datetime value | Datetime functions | DATETIME() | () | The SAS datetime value at the moment of execution, in the form NNNNNNNNN.N |
| Interval Tasks | | | | |
| Return the number of specified time intervals that lie between the two date or datetime values | Interval functions | INTCK | week2<br>01aug60<br>01jan13 | 1368 |
| Advances a date, time, or datetime value by a given interval, and returns a date, time, or datetime value | Interval functions | INTNX | day<br>17mar12<br>365 | 19434 |

1 In SAS, a Julian date is a date in the form YYNNN or YYYYNNN, where YY is a two-digit year, YYYY is a four-digit year, and NNN is the ordinal offset from January 1 of the year YY or YYYY. SAS processes Julian dates only for valid SAS dates.

SAS also supports these formats and informats:

- ISO 8601 basic and extended forms for dates, times, datetimes, durations, intervals, and time zones. For more information, see “Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 45 and “Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 540.
- There are international formats and informats that are equivalent to some of the most commonly used English-language date formats and informats. For more information, see “Dictionary of Formats” and “Dictionary of Informats”.

# Examples

## Example 1: Displaying Date, Time, and Datetime Values as Recognizable Dates and Times

The following example demonstrates how a value might be displayed as a date, a time, or a datetime. Remember to select the SAS language element that converts a SAS date, time, or datetime value to the intended date, time, or datetime format. See the previous tables for examples.

**Note:**

- Time formats count the number of seconds within a day, so the values are between 0 and 86400.
- DATETIME formats count the number of seconds since January 1, 1960. For datetimes that are greater than 02JAN1960:00:00:01 (integer of 86401), the datetime value is always greater than the time value.
- When in doubt, look at the contents of your data set for clues as to which type of value you are dealing with.

This program uses the DATETIME, DATE, and TIMEAMPM formats to display the value 86399 to a date and time, a calendar date, and a time.

```
options nodate;
data test;
   Time1=86399;
   format Time1 datetime.;
   Date1=86399;
   format Date1 date9.;
   Time2=86399;
   format Time2 timeampm.;
run;
proc print data=test;
   title  'Same Number, Different SAS Values';
   footnote1 'Time1 is a SAS DATETIME value';
   footnote2 'Date1 is a SAS DATE value';
   footnote3 'Time2 is a SAS TIME value';
run;
footnote;
```

***Output 1.2*** *Datetime, Date, and Time Values for 86399*

**Same Number, Different SAS Values**

| Obs | Time1 | Date1 | Time2 |
|---|---|---|---|
| 1 | 01JAN60:23:59:59 | 20JUL2196 | 11:59:59 PM |

Time1 is a SAS DATETIME value
Date1 is a SAS DATE value
Time2 is a SAS TIME value

# Example 2: Reading, Writing, and Calculating Date Values

This program reads four regional meeting dates and calculates the dates on which announcements should be mailed.

```
data meeting;

   input region $ mtg : mmddyy8.;
   sendmail=mtg-45;
   datalines;
N  11-24-12
S  12-28-12
E  12-03-12
W  10-04-12
;

proc print data=meeting;
   format mtg sendmail date9.;
   title 'When To Send Announcements';
run;
```

***Output 1.3*** *Calculated Date Values: When to Send Mail*

**When To Send Announcements**

| Obs | region | mtg | sendmail |
|---|---|---|---|
| 1 | N | 24NOV2012 | 10OCT2012 |
| 2 | S | 28DEC2012 | 13NOV2012 |
| 3 | E | 03DEC2012 | 19OCT2012 |
| 4 | W | 04OCT2012 | 20AUG2012 |

# About Date and Time Intervals

## Definitions

duration
: is an integer that represents the difference between any two dates or times or datetimes. Date durations are integer values that represent the difference, in the number of days, between two SAS dates. Time durations are decimal values that represent the number of seconds between two times or datetimes.

> **TIP** Date and datetime durations can be easily calculated by subtracting the smaller date or datetime from the larger. When dealing with SAS times, special care must be taken if the beginning and the end of a duration are on different calendar days. Whenever possible, the simplest solution is to use datetimes rather than times.

interval
: is a unit of measurement that SAS can count within an elapsed period of time, such as DAYS, MONTHS, or HOURS. SAS determines date and time intervals based on fixed points on the calendar, the clock, or both. The starting point of an interval calculation defaults to the beginning of the period in which the beginning value falls, which might not be the actual beginning value that is specified. For example, if you are using the INTCK function to count the months between two dates, regardless of the actual day of the month that is specified by the date in the beginning value, SAS treats the date as the first of that month.

## Syntax

SAS provides date, time, and datetime intervals for counting different periods of elapsed time. You can create multiples of the intervals and shift their starting point. Use intervals with the INTCK and INTNX functions and with procedures that support numbered lists (such as the PLOT procedure). This is the form of an interval:

*name<multiple><.starting-point>*

The terms in an interval have the following definitions:

*name*
: is the name of the interval. For a list of intervals and their definitions, see the following table.

*multiple*
: creates a multiple of the interval. *multiple* can be any positive number. The default is 1. For example, YEAR2 indicates a two-year interval.

.starting-point
is the starting point of the interval. By default, the starting point is 1. A value greater than 1 shifts the start to a later point within the interval. The unit for shifting depends on the interval, as shown in the following table. For example, YEAR.3 specifies a yearly period from the first of March through the end of February of the following year.

# Intervals by Category

***Table 1.3*** *Intervals Used with Date and Time Functions*

| Category | Interval | Definition | Default Starting Point | Shift Period | Example | Description |
|---|---|---|---|---|---|---|
| Date | DAY | Daily intervals | Each day | Days | DAY3 | Three-day intervals starting on Sunday |
| | WEEK | Weekly intervals of seven days | Each Sunday | Days (1=Sunday … 7=Saturday) | WEEK.7 | Weekly with Saturday as the first day of the week |
| | WEEKDAY <*days*W> | Daily intervals with Friday-Saturday-Sunday | Each day | Days | WEEKDAY1W | Six-day week with Sunday as a weekend day |
| | | Counted as the same day (five-day work week with a Saturday-Sunday weekend). *days* identifies the weekend days by number (1=Sunday .. . 7=Saturday). By default, *days*=17. | | | WEEKDAY35W | Five-day week with Tuesday and Thursday as weekend days (W indicates that day 3 and day 5 are weekend days) |
| | TENDAY | Ten-day intervals (a | First, eleventh, | Ten-day periods | TENDAY4.2 | Four ten-day periods |

| Category | Interval | Definition | Default Starting Point | Shift Period | Example | Description |
|---|---|---|---|---|---|---|
| | | U.S. automobile industry convention) | and twenty-first of each month | | | starting at the second TENDAY period |
| | SEMIMONTH | Half-month intervals | First and sixteenth of each month | Semi-monthly periods | SEMIMONTH2.2 | Intervals from the 16th of one month through the 15th of the next month |
| | MONTH | Monthly intervals | First of each month | Months | MONTH2.2 | February-March, April-May, June-July, August-September, October-November, and December-January of the following year |
| | QTR | Quarterly (three-month) intervals | January 1<br>April 1<br>July 1<br>October 1 | Months | QTR3.2 | Three-month intervals starting on April 1, July 1, October 1, and January 1 |
| | SEMIYEAR | Semiannual (six-month) intervals | January 1<br>July 1 | Months | SEMIYEAR.3 | Six-month intervals, March-August, and September-February |
| | YEAR | Yearly intervals | January 1 | Months | | |
| Datetime | Add DT to any of the date intervals | Interval that corresponds to the associated date interval | Midnight of January 1, 1960 | | DTMONTH<br>DTWEEKDAY | |

| Category | Interval | Definition | Default Starting Point | Shift Period | Example | Description |
|---|---|---|---|---|---|---|
| Time | SECOND | Second intervals | Start of the day (midnight) | Seconds | | |
| | MINUTE | Minute intervals | Start of the day (midnight) | Minutes | | |
| | HOUR | Hourly intervals | Start of the day (midnight) | Hours | | |

# Example: Calculating a Duration

This program reads the project start and end dates. Then, the program calculates the duration between the dates.

```
options nodate pageno=1 linesize=80 pagesize=60;

data projects;
   input Projid @5 startdate date9. @15 enddate date9.;
   Duration=enddate-startdate;
   datalines;
398 17oct1997 02nov1997
942 22jan1998 11mar1998
167 15dec1999 15feb2000
250 04jan2001 11jan2001
;

proc print data=projects;
   format startdate enddate date9.;
      title 'Days Between Project Start and Project End';
run;
```

***Output 1.4*** *Calculating the Duration between Start and End Dates*

**Days Between Project Start and Project End**

| Obs | Projid | startdate | enddate | Duration |
|---|---|---|---|---|
| 1 | 398 | 17OCT1997 | 02NOV1997 | 16 |
| 2 | 942 | 22JAN1998 | 11MAR1998 | 48 |
| 3 | 167 | 15DEC1999 | 15FEB2000 | 62 |
| 4 | 250 | 04JAN2001 | 11JAN2001 | 7 |

# Boundaries of Intervals

SAS associates date and time intervals with fixed points on the calendar. For example, the MONTH interval represents the time from the beginning of one calendar month to the next, not a period of 30 or 31 days. When you use date and time intervals (for example, with the INTCK or INTNX functions), SAS bases its calculations on the calendar divisions that are present. Consider the following examples:

***Table 1.4*** *Using INTCK and INTNX*

| Example | Result | Explanation |
|---|---|---|
| `mnthnum1=intck( 'month', '25aug2000'd, '05sep2000'd);` | mnthnum1=1 | The number of MONTH intervals the INTCK function counts depends on whether the first day of a month falls within the period. |
| `mnthnum2=intck( 'month', '01aug2000'd, '31aug2000'd);` | mnthnum2=0 | |
| `next=intnx('month', '25aug2000'd,1);` | next represents 01sep2000 | The INTNX function produces the SAS date value that corresponds to the beginning of the next interval. |

**Note:** The only intervals that do not begin on the same date in each year are WEEK and WEEKDAY. A Sunday can occur on any date because the year is not divided evenly into weeks.

# Single-Unit Intervals

Single-unit intervals begin at the following points on the calendar:

***Table 1.5*** *Single-Unit Intervals*

| Single-Unit Interval | Beginning Point on the Calendar |
|---|---|
| DAY | Each day |
| WEEKDAY | For a standard weekday<br>■ *Start day–End day*<br>■ Monday–Monday |

| Single-Unit Interval | Beginning Point on the Calendar |
|---|---|
| | ■ Tuesday–Tuesday<br>■ Wednesday–Wednesday<br>■ Thursday–Thursday<br>■ Friday–Sunday |
| WEEK | Each Sunday |
| TENDAY | The first, eleventh, and twenty-first of each month |
| SEMIMONTH | The first and sixteenth of each month |
| MONTH | The first of each month |
| QTR | The first of January, April, July, and October |
| SEMIYEAR | The first of January and July |
| YEAR | The first of January |

Single-unit time intervals begin as follows:

***Table 1.6*** *Single-Unit Time Intervals*

| Single-Unit Time Interval | Beginning Point |
|---|---|
| SECOND | Each second |
| MINUTE | Each minute |
| HOUR | Each hour |

# Multi-unit Intervals

## Multi-unit Intervals Other Than Multi-week Intervals

Multi-unit intervals such as MONTH2 or DAY50 also depend on calendar measures, but they introduce a new problem: SAS can find the beginning of a unit (for example, the first of a month), but where does that unit fall in the interval? For

example, does the first of October mark the first or the second month in a two-month interval?

For all multi-unit intervals except multi-week intervals, SAS creates an interval that begins on January 1, 1960, and counts forward from that date to determine where individual intervals begin on the calendar. As a practical matter, when a year can be divided evenly by an interval, think of the intervals as beginning with the current year. Thus, MONTH2 intervals begin with January, March, May, July, September, and November. Consider this example:

***Table 1.7*** *Month2 Intervals*

| SAS Statement | Result |
|---|---|
| `howmany1=intck('month2','15feb2000'd, '15mar2000'd);` | howmany1=1 |
| `count=intck('day50','01oct1998'd, '01jan1999'd);` | count=1 |

In the preceding example, SAS counts 50 days beginning with January 1, 1960,then another 50 days, and so on. As part of this count, SAS counts one DAY50 interval between October 1, 1998, and January 1, 1999. For example, to determine the date on which the next DAY50 interval begins, use the INTNX function:

***Table 1.8*** *Using the INTNX Function*

| SAS Statement | Result |
|---|---|
| `start=intnx('day50','01oct98'd,1);` | SAS date value 14200, or Nov 17, 1998 |

The next interval begins on November 17, 1998.

Time intervals (those that represent divisions of a day) are aligned with the start of the day; that is, midnight. For example, HOUR8 intervals divide the day into the periods 00:00 to 08:00, 8:00 to 16:00, and 16:00 to 24:00 (the next midnight).

# Multi-week Intervals

Multi-week intervals such as WEEK2 present a special case. In general, weekly intervals begin on Sunday, and SAS counts a week whenever it passes a Sunday. However, SAS cannot calculate multi-week intervals based on January 1, 1960, because that date fell on a Friday, as shown here:

***Figure 1.3*** *Calculating Multi-week Intervals*

| Dec | Su | Mo | Tu | We | Th | Fr | Sa | Jan |
|---|---|---|---|---|---|---|---|---|
| 1959 | 27 | 28 | 29 | 30 | 31 | 1 | 2 | 1960 |

Therefore, SAS begins the first interval on Sunday of the week containing January 1, 1960;that is, on Sunday, December 27, 1959. SAS counts multi-week intervals from that point. The following example counts the number of two-week intervals in the month of August 1998:

***Table 1.9*** *Counting Two-Week Intervals*

| SAS Statement | Result |
|---|---|
| `count=intck('week2','01aug98'D, '31aug98'D);` | count=3 |

To see the beginning date of the next interval, use the INTNX function, as shown here:

***Table 1.10*** *Using INTNX to See the Beginning Date of an Interval*

| SAS Statement | Result |
|---|---|
| `begin=intnx('week2','01aug1998'd,1);` | “Begin” represents SAS date 14093 or August 02, 1998. |

The next interval begins on August 16.

# Shifted Intervals

## Using Shifted Intervals

Shifting the beginning point of an interval is useful when you want to make the interval represent a period in your data. For example, if your company's fiscal year begins on July 1, you can create a year beginning in July by specifying the YEAR.7 interval. Similarly, you can create a period matching U.S. presidential elections by specifying the YEAR4.11 interval. This section discusses how to use shifted intervals and how SAS creates them.

## How to Use Shifted Intervals

When you shift a time interval by a subperiod, the shift value must be less than or equal to the number of subperiods in the interval. For example, YEAR.12 is valid (yearly periods beginning in December), but YEAR.13 is not. Similarly, YEAR2.25 is not valid because there is no 25th month in the two-year period.

In addition, you cannot shift an interval by itself. For example, you cannot shift the interval MONTH because the shifting subperiod for MONTH is one month and MONTH contains only one monthly subperiod. However, you can shift multi-unit

intervals by the subperiod. For example, MONTH2.2 specifies bimonthly periods starting on the first day of the second month.

## How SAS Creates Shifted Intervals

For all intervals except those based on weeks, SAS creates shifted intervals by creating the interval based on January 1, 1960, by moving forward the required number of subperiods and by counting shifted intervals from that point. For example, suppose you create a shifted interval called DAY50.5. SAS creates a 50-day interval in which January 1, 1960, is day 1. SAS then moves forward to day 5. (Note that the difference, or amount of movement, is four days.) SAS begins counting shifted intervals from that point. The INTNX function demonstrates that the next interval begins on January 5, 1960:

***Table 1.11*** *Using INTNX to Determine When an Interval Begins*

| SAS Statement | Result |
|---|---|
| `start=intnx('day50.5','01jan1960'd,1);` | SAS date value 4, or Jan 5, 1960 |

For shifted intervals based on weeks, SAS first creates an interval based on Sunday of the week containing January 1, 1960 (that is, December 27, 1959). Then, SAS moves forward the required number of days. For example, suppose you want to create the interval WEEK2.8 (biweekly periods beginning on the second Sunday of the period). SAS measures a two-week interval based on Sunday of the week containing January 1, 1960, and begins counting shifted intervals on the eighth day of that. The INTNX function shows the beginning of the next interval.

***Table 1.12*** *Using the INTNX Function to Show the Beginning of the Next Interval*

| SAS Statement | Result |
|---|---|
| `start=intnx('week2.8','01jan1960'd,1);` | SAS date value 2, or Jan 3, 1960 |

You can also shift time intervals. For example, HOUR8.7 intervals divide the day into the periods 06:00 to 14:00, 14:00 to 22:00, and 22:00 to 06:00.

# Custom Intervals

You can define custom intervals and associate interval data sets with new interval names when you use the INTERVALDS= system option. An interval name cannot be a reserved SAS name. The dates for these intervals are located in a SAS data set that you create. The data set must contain the variable Begin. For each observation, the Begin variable represents the start of an interval. You can specify a second variable, End, to represent the end of the interval, but it is not required. If the End variable is not present in the data set, the end of an interval is inferred by the next Begin variable value. After the custom intervals have been defined, you can

use them with the INTCK and INTNX functions just as you would use standard intervals.

The INTERVALDS= system option enables you to increase the number of allowable intervals. In addition to the standard list of intervals (DAY, WEEKDAY, and so on), the names that are listed in INTERVALDS= are valid as well.

**Note:** Nested custom intervals are not supported.

# Retail Calendar Intervals: ISO 8601 Compliant

The retail industry often accounts for its data by dividing the yearly calendar into four 13-week periods, based on one of the following formats: 4-4-5, 4-5-4, and 5-4-4. The first, second, and third numbers specify the number of weeks in the first, second, and third month of each period, respectively. Retail calendar intervals facilitate comparisons across years, because week definitions remain consistent from year to year.

The intervals that are created from the formats can be used in any of the following functions: INTCINDEX, INTCK, INTCYCLE, INTFIT, INTFMT, INTGET, INTINDEX, INTNX, INTSEAS, INTSHIFT, and INTTEST.

The following table lists calendar intervals that are used in the retail industry and that are ISO 8601 compliant.

***Table 1.13*** *Calendar Intervals Used in the Retail Industry*

| Interval | Description |
|---|---|
| YEARV | Specifies ISO 8601 yearly intervals. The ISO 8601 year begins on the Monday on or immediately preceding January 4. Note that it is possible for the ISO 8601 year to begin in December of the preceding year. Also, some ISO 8601 years contain a leap week. The beginning subperiod *s* is written in ISO 8601 weeks (WEEKV). |
| R445YR | Is the same as YEARV except that in the retail industry the beginning subperiod *s* is 4-4-5 months (R445MON). |
| R454YR | Is the same as YEARV except that in the retail industry the beginning subperiod *s* is 4-5-4 months (R454MON). |
| R544YR | Is the same as YEARV except that in the retail industry the beginning subperiod *s* is 5-4-4 months (R544MON). |
| R445QTR | Specifies retail 4-4-5 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The beginning subperiod *s* is 4-4-5 months (R445MON). |

| Interval | Description |
|---|---|
| R454QTR | Specifies retail 4-5-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The beginning subperiod *s* is 4-5-4 months (R454MON). |
| R544QTR | Specifies retail 5-4-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The beginning subperiod *s* is 5-4-4 months (R544MON). |
| R445MON | Specifies retail 4-4-5 monthly intervals. The 3rd, 6th, 9th, and 12th months are five ISO 8601 weeks long with the exception that some 12th months contain leap weeks. All other months are four ISO 8601 weeks long. R445MON intervals begin with the 1st, 5th, 9th, 14th, 18th, 22nd, 27th, 31st, 35th, 40th, 44th, and 48th weeks of the ISO year. The beginning subperiod *s* is 4-4-5 months (R445MON). |
| R454MON | Specifies retail 4-5-4 monthly intervals. The 2nd, 5th, 8th, and 11th months are five ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R454MON intervals begin with the 1st, 5th, 10th, 14th, 18th, 23rd, 27th, 31st, 36th, 40th, 44th, and 49th weeks of the ISO year. The beginning subperiod *s* is 4-5-4 months (R454MON). |
| R544MON | Specifies retail 5-4-4 monthly intervals. The 1st, 4th, 7th, and 10th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R544MON intervals begin with the 1st, 6th, 10th, 14th, 19th, 23rd, 27th, 32nd, 36th, 40th, 45th, and 49th weeks of the ISO year. The beginning subperiod *s* is 5-4-4 months (R544MON). |
| WEEKV | Specifies ISO 8601 weekly intervals of seven days. Each week begins on Monday. The beginning subperiod *s* is calculated in days (DAY). Note that WEEKV differs from WEEK in that WEEKV.1 begins on Monday, WEEKV.2 begins on Tuesday, and so on. |

# 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: 1 byte, 2 bytes, or 4 bytes. 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 platforms that are based on Intel (little endian). In SAS, 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, OpenVMS on HP Integrity Servers, 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, 2 bytes as 00 01, and 4 bytes as 00 00 00 01. On little endian platforms, the value 1 is stored in 1 byte as 01 (the same as big endian), in 2 bytes as 01 00, and in 4 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 is set on 1. For example, –2 would be represented in 1, 2, and 4 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 notation.

## Writing Data Generated on Big Endian and Little Endian Platforms

SAS can read signed and unsigned integers regardless of whether they were generated on a big endian or a little endian platform. 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 8 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.

| 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) |

| Language | 2 Bytes | 4 Bytes |
|---|---|---|
| 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, which results in a table of 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 typically 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 SAS is transcoding the ASCII dollar sign to the Danish EBCDIC dollar sign, the hexadecimal representation for the character is converted from the value 24 to the value 67.

To learn the encoding of a particular SAS data set for SAS 9 and later:

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.

Here are several situations in which data might commonly be transcoded:

- 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 that are running in different locales

For more information about SAS features that are 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 that involve 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 1 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, the number is negative. Otherwise, the number is positive.

The following information 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 handle the representation. The representation is consistent between ASCII and EBCDIC platforms.
- The S370FPDU format and informat expect to have an F in the last nibble. A packed decimal expects no sign nibble.

## Zoned Decimal Data

The following information applies to zoned decimal data representation:

- A zoned decimal representation stores a decimal digit in the low-order nibble of each byte. For all bytes except the byte that contains 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 expect 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 2 bytes. The first byte contains the hexadecimal value F1, and the second byte contains the hexadecimal value C5.

## Packed Julian Dates

The following information 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, whereas 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 two-digit or four-digit years. If you use two-digit years, SAS uses the setting of the YEARCUTOFF= system option to determine the true year.

# Platforms That Support Packed Decimal and Zoned Decimal Data

Some platforms have native instructions to support packed and zoned decimal data. Other platforms must use software to emulate the computations. For example, the

IBM mainframe has an Add Pack instruction to add packed decimal data, but the platforms that are based on Intel have no such instruction and must convert the decimal data to some other format.

# Languages That Support 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 versus COBOL II Clauses | Corresponding S370Fxxx Formats and Informats |
|---|---|
| PIC S9(X) PACKED-DECIMAL | S370FPD*w*. |
| PIC 9(X) PACKED-DECIMAL | S370FPDU*w*. |
| PIC S9(W) DISPLAY | S370FZD*w*. |
| PIC 9(W) DISPLAY | S370FZDU*w*. |
| PIC S9(W) DISPLAY SIGN LEADING | S370FZDL*w*. |
| PIC S9(W) DISPLAY SIGN LEADING SEPARATE | S370FZDS*w*. |
| PIC S9(W) DISPLAY SIGN TRAILING SEPARATE | S370FZDT*w*. |

For the packed decimal representation listed in the preceding table, 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 3 bytes, and the value of W is 3.

You can substitute COMP-3 for PACKED-DECIMAL.

In IBM assembly language, the P directive indicates packed decimal data, and the Z directive indicates zoned decimal data. Here is an excerpt from an assembly language list that shows 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 C language.

# 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 types of data representation for these formats and informats. The formats and informats that begin with S370 refer to IBM mainframe representation.

| Format | Type of Data Representation | Corresponding Informat | Comment |
|---|---|---|---|
| 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 (hexadecimal 40) to EBCDIC zero (hexadecimal F0); corresponds to the informat as zoned decimal. |
| none | Zoned decimal | ZDV | Zoned decimal notation other than IBM. |
| 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 |

| Format | Type of Data Representation | Corresponding Informat | Comment |
|---|---|---|---|
| | | | (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. |
| none | Packed decimal | SHRSTAMP | Input layout is *yyyyddd*F*hhmmssth*, where *yyyyddd*F is the packed Julian date; *yyyy* is a zero-based year from 1900. |
| none | Packed decimal | SMFSTAMP | Input layout is *xxxxxxxxyyyyddd*F, where *yyyyddd*F is the packed Julian date; *yyyy* is a zero-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 zero-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 values of the ISO 8601 dates, time, datetime, durations, and interval:

***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* uses 1–3 digits for values read by the $N8601B informat and the $N8601E informat.

*ffffff* use 1–6 digits for informats other than the $N8601B and $N8601E informats.

**Y**
indicates that a year value precedes this character in a duration.

**M**
as part of a date, indicates that a month value precedes this character in a duration.

**D**
indicates that a day value precedes this character in a duration.

**H**
indicates that an hour value precedes this character in a duration.

**M**
as part of a time, indicates that a minute value precedes this character in a duration.

**S**
indicates that a seconds value precedes this character in a duration.

# Definitions

Local time
is the local time when a time zone is not specified by the TIMEZONE= system option.

Time zone offset
specifies the number of hours and minutes that a time zone is from Universal Coordinate Time (UTC) in the form +|–*hh:mm* or +|–*hhmm*.

User local time
is the local time for the time zone that is specified by the TIMEZONE= system option. For more information, see "TIMEZONE=" in *SAS System Options: Reference*.

UTC
Universal Coordinate Time is the time at the zero meridian, near Greenwich, England. UTC is a datetime value that uses the ISO 8601 basic form *yyyymmdd*T*hhmmss*+|–*hhmm* or the ISO 8601 extended form *yyyy-mm-dd*T*hh:mm:ss*+|–*hh:mm*.

# About Dates and Times That Use the ISO 8601 Basic and Extended Notations

ISO 8601 is an international standard for representing dates and time, including many variations for representing dates, times, and intervals. The two main representations of date, time, and datetime values within the ISO 8601 standards are the basic and extended notations. A value is considered extended when delimiters separate the various components within the value, whereas a basic value omits the delimiters. The extended format requires hyphen delimiters for date components (year, month, and day) and colon delimiters for time components (hour, minute, and second). Spaces are not allowed in any ISO 8601 representation. The structures for each data type require that you fill each placeholder with a value, including adding a zero to single-digit months, days, hours, and minutes. When you specify a datetime value, an uppercase T is the required delimiter between the date and time.

Some of the ISO 8601 formats are for formatting time and datetime values in UTC. The time or datetime value includes a time zone offset, which is a positive or negative number that represents the number of hours a time zone is from the zero meridian. Positive numbers are east of the zero meridian, and negative numbers are west of the zero meridian. The time zone offset at the zero meridian is always zero.

Here are examples of basic and extended notations for ISO 8601 date, time, datetime, and duration values:

- 2018-02-01 is a date that is formatted using the extended notation.
- 125234-0500 is a time with a time zone offset five hours west of the zero meridian and is formatted using the basic notation.
- 2018-02-01T12:52:34+09:00 is a datetime value with a time zone offset nine hours east of the zero meridian and is formatted using the extended notation.
- P20180501T120000 is a duration value that is formatted using the basic notation.

When SAS reads an ISO 8601 value that specifies UTC with a time offset, the time or datetime value is adjusted to account for the offset from the zero meridian.

The SAS ISO 8601 formats for UTC with a time zone offset are based on the following time, datetime, and time zone offsets:

- the zero meridian time or datetime near Greenwich, England. (The offset is always +|–0000 or +|–00:00.)
- the local time or datetime, which uses the zero meridian time with a time zone offset for the local time.
- a zero meridian datetime that uses a time zone offset for the user local time.
- a user local time or datetime, which uses the current time for a time zone with a time zone offset for the user local time.

# Basic ISO 8601 Date, Time, and Datetime Values

Basic formats and informats are prefixed with B8601 and take these forms:

| | |
|---|---|
| Date | *yyyymmdd* |
| Time | *hhmmss<ffffff>* |
| Datetime | *yyyymmdd*T*hhmmss<ffffff>* |
| Time with time zone | *hhmmss<ffffff>*+\|–*hhmm* |
| Datetime with time zone | *yyyymmdd*T*hhmmss<ffffff>*+\|–*hhmm* |

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

| Date, Time, or Datetime | ISO 8601 Notation | Example | Format |
|---|---|---|---|
| Date | *yyyymmdd* | 20180915 | B8601DAw. on page 146 |
| Writes the date from a datetime. | *yyyymmdd* | 20180915 | B8601DNw. on page 148 |
| Time | *hhmmssffffff* | 155300322348 | B8601TMw.d on page 158 |
| Zero meridian time and time zone offset.<br>The time zone offset is always +0000 or Z. | *hhmmss*+\|–*hhmm* | 155300+0000 | B8601TZw. on page 161 |
| | *hhmmss*Z | 155300Z | B8601TZw. on page 161 |
| Zero meridian time that uses a time zone offset for the local time. | *hhmmss*+\|–*hhmm* | 155300–0500 | B8601LZw. on page 156 |
| Converts time to a user's local time by using a time zone offset for the user local time. | *hhmmss*+\|–*hhmm* | 155300–0500 | B8601TXw. on page 159 |
| Datetime | *yyyymmdd*T*hhmmssffffff* | 20180915T155300 | B8601DTw.d on page 149 |
| Zero meridian datetime and time zone offset. | *yyyymmdd*T*hhmmss*+\|–*hhmm* | 20180915T155300–0000 | B8601DZw. on page 153 |

| Date, Time, or Datetime | ISO 8601 Notation | Example | Format |
|---|---|---|---|
| The time zone offset is always +0000 or Z. | | | |
| | *yyyymmdd*T*hhmmss*Z | 20180915T155300Z | B8601DZw. on page 153 |
| Zero meridian datetime that uses a time zone offset for the user local time. | *yyyymmdd*T*hhmmss*+\|–*hhmm* | 20180915T155300+0500 | B8601LXw. on page 154 |
| Converts a datetime value to the user local time by using a time zone offset for the user local time. | *yyyymmdd*T*hhmmss*+\|–*hhmm* | 20180915T155300+0500 | B8601DXw. on page 151 |

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

# Extended ISO 8601 Date, Time, and Datetime Values

Extended formats and informats are prefixed with E8601 and take these forms:

| | |
|---|---|
| Date | *yyyy-mm-dd* |
| Time | *hh:mm:ss<ffffff>* |
| Datetime | *yyyy-mm-dd*T*hh:mm:ss<ffffff>* |
| Time with time zone | *hh:mm:ss<ffffff>*+\|–*hh:mm* or *hh:mm:ss<ffffff>*Z |
| Datetime with time zone | *yyyy-mm-dd*T*hh:mm:ss<ffffff>*+\|–*hh:mm* or *yyyy-mm-dd*T*hh:mm:ss<ffffff>*Z |

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

| Date, Time, or Datetime | ISO 8601 Notation | Example | Format |
|---|---|---|---|
| Date | *yyyy-mm-dd* | 2018-09-15 | E8601DAw. on page 193 |
| Writes the date from a datetime. | *yyyy-mm-dd* | 2018-09-15 | E8601DNw. on page 148 |

| Date, Time, or Datetime | ISO 8601 Notation | Example | Format |
|---|---|---|---|
| Time | *hh:mm:ss.ffffff* | 15:53:00.322348 | E8601TMw.d on page 158 |
| Zero meridian time and time zone offset.<br>The time zone offset is always +00:00. | *hh:mm:ss.ffffff*+\|–*hh:mm* | 15:53:00+00:00 | E8601TZw.d on page 161 |
| Zero meridian time that uses a local time zone offset. | *hh:mm:ss.ffffff*+\|–*hh:mm* | 15:53:00+05:00 | E8601LZw.d on page 156 |
| Converts time to user local time by using a local time zone offset. | *hh:mm:ss*+\|–*hh:mm* | 15:53:00+05:00 | EB8601TXw. on page 206 |
| Datetime | *yyyy-mm-dd*T*hh:mm:ss.ffffff* | 2018-09-15T15:53:00 | E8601DTw.d on page 149 |
| Zero meridian datetime and time zone offset.<br>The time zone offset is always +00:00. | *yyyy-mm-dd*T*hh:mm:ss.nnnnnn* +\|–*hh:mm* | 2018-09-15T15:53:00+00:00 | E8601DZw.d on page 199 |
| Zero meridian datetime that uses a time zone offset for the user local time. | *yyyy-mm-dd*T*hh:mm:ss* +\|–*hh:mm* | 2018-09-15T15:53:00–05:00 | E8601LXw. on page 201 |
| Converts a datetime value to the user local time by using a time zone offset for the user local time. | *yyyy-mm-dd*T*hh:mm:ss* +\|–*hh:mm* | 2018-09-15T185300–0500 | E8601DXw. on page 197 |

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

For more information about time zones, see "Specifying Time Zones in SAS" in *SAS National Language Support (NLS): Reference Guide* and "TIMEZONE=" in *SAS System Options: Reference*.

# Tips for Remembering UTC Formats That Use Time Zone Offsets

Here are tips to help you remember which format to use for UTC:

- A T in the last two letters is a time value.
- A D in the last two letters is a datetime value.
- A Z in the last two letters is a zero meridian time and a zero meridian offset, except for LZ.
- An L in the last two letters is a zero meridian time with a local or user local time zone offset.
- An X in the last two letters is for user local time or datetime and uses a user local time zone offset that is determined by the TIMEZONE= system option.

Here is information to help you determine the format for UTC:

| | TZ or DZ | LZ | LX | TX or DX |
|---|---|---|---|---|
| Zero meridian date or datetime that uses a zero meridian time zone offset | ✓ | | | |
| Zero meridian time that uses a local time zone offset | | ✓ | | |
| Zero meridian datetime that uses a user local time zone offset | | | ✓ | |
| User local time or datetime that uses a user local time zone offset | | | | ✓ |

# Examples of Reading and Writing Basic and Extended ISO 8601 Date, Time, and Datetime Values

## About the Basic and Extended ISO 8601 Examples

The examples in this section demonstrate how to use various informats to read date, time, and datetime values into SAS date, time, and datetime variables. The examples also illustrate how to use formats to write these values in a way that is meaningful to users.

# Comparing ISO 8601 Extended Format Output

This example compares the output for the different extended notations for time and datetime values.

**Note:** The E8601DX, E8601LX, and E8601TX formats that are used in the example are not supported in CAS.

```
data _null_;
d='15Sep2018:5:53:00'dt;
tm='05:53:00't;
put 'd=' d datetime.;
put 'e8601dz=' d e8601dz.;
put 'e8601lx=' d e8601lx.;
put 'e8601dx=' d e8601dx.;
put 'tm=' tm time.;
put 'e8601tz=' tm e8601tz.;
put 'e8601lz=' tm e8601lz.;
put 'e8601tx=' tm e8601tx.;
run;
```

The program executed using the local time for the eastern United States and no value for the TIMEZONE= system option. Therefore, the time zone formats E8601LZ., E8601DX., and E8601TX. show local times.

- The output for the E8601DZ. and E8601TZ. formats writes a SAS datetime and time value as the time at the zero meridian by using a time zone offset of +0000 or +00:00.
- The output for the E8601LX. and E8601LZ. formats writes a SAS datetime value as the time at the zero meridian by using a time zone offset for the local time.
- The output for the E8601DX. and E8601TX. formats converts the time to Eastern Time by using a time zone offset for the local time.

```
d=15SEP13:05:53:00
e8601dz= 2018-09-15T05:53:00+00:00
e8601lx= 2018-09-15T05:53:00-05:00
e8601dx= 2018-09-15T00:53:00-05:00
tm= 5:53:00
e8601tz=05:53:00+00:00
e8601lz=05:53:00-05:00
e8601tx=00:53:00-05:00
```

Here are the results when the TIMEZONE= option is set to America/Los_Angeles:

```
d=15SEP13:05:53:00
e8601dz= 2018-09-15T05:53:00+00:00
e8601lx= 2018-09-15T05:53:00-07:00
e8601dx= 2018-09-14T22:53:00-07:00
tm= 5:53:00
e8601tz=05:53:00+00:00
e8601lz=05:53:00-05:00
e8601tx=21:53:00-08:00
```

- The E8601DZ. and E8601TZ. formats do not change. They always show the time as the time at the zero meridian.
- The E8601LX. format shows the zero meridian time by using the time zone offset for the America/Los_Angeles time zone.
- The E8601DX. formats show the local date and time in Los Angeles by using the time zone offset for the America/Los_Angeles time zone. The Los Angeles time zone offset of –07:00 indicates that the local time is seven hours earlier than the time at the zero meridian. This changes the date to the 14th.
- The E8601LZ. format shows the time at the zero meridian by using the time zone offset for the eastern United States because that is where the code executed.
- The E8601TX. format shows the time seven hours earlier from the zero meridian by using a time zone offset for the America/Los_Angeles time zone.

# Reading and Writing Date Values

Suppose that you have a clinical trial where an event begins on April 2, 2018, and ends on April 8, 2018. The dates are recorded without time values, as follows: 20180402 and 2018-04-08. You can read these values into SAS with the B8601DA*w*. and E8601DA*w*. informats. You can write dates in the same form by using the B8601DA*w*. and E8601DA*w*. formats. These formats write the newly created SAS dates in an easy-to-read layout rather than the numeric value of days since 1/1/1960.

```
data a;
input var1 b8601da8. +1 var2 e8601da10.;
put var1=b8601da. var2=e8601da.;
datalines;
20180402 2018-04-08
;
run;
```

Here is the output from the SAS log:

```
var1=20180402 var2=2018-04-08
```

By using the B8601DN*w*. and E8601DN*w*. formats, you can extrapolate the date from a datetime value. This example reads the datetime value by using the B8601DN*w*. informat, and writes the date by using the B8601DN*w*. format:

```
data _null_;
input @1 dt b8601dn.;
put dt b8601dn.;
datalines;
```

```
20180915T094322
run;
```

Here is the output from the SAS log:

```
20180915
```

# Reading and Writing Time Values with No Time Zone Offsets

You can read time values that do not have time zone offset values into SAS time values by using the B8601TM*w.d* and E8601TM*w.d* informats. The B8601TM*w.d* and the E8601TM*w.d* formats write time values that do not have time zone offset values, as shown in this example:

```
data _null_;
x=input('12:34:56',e8601tm8.);
put x=b8601tm8. x=e8601tm10.;
run;
```

Here is the output from the SAS log:

```
x=123456 x=12:34:56
```

# Reading and Writing Time Values with Time Zone Offsets

This example uses the E8601TZ*w.d* informat to read a time value that contains a time zone offset. The B8601TZ*w.d* and E8601TZ*w.d* formats write the time values by using a time zone offset for the zero meridian:

```
data _null_;
x=input('12:34:56-04:00',e8601tz14.);
put x=e8601tz14.;
put x=b8601tz.;
run;
```

Here is the output from the SAS log:

```
x=16:34:56+00:00
x=163456+0000
```

You can adjust a time to be the time in another time zone by using a time zone offset. You specify the time zone by using the TIMEZONE= system option, and then format the time by using the B8601TX*w*. or E8601TX*w*. format. This example writes the time for a user by using the time zone name PST (Pacific Time):

```
options timezone=pst;
```

```
data _null_;
x='12:34:56't;
put x=e8601tx.;
run;
```

Here is the output from the SAS log:

```
04:34:56-08:00
```

# Reading and Writing Time Values with Local Time Zone Offsets

Because time values are scalar, SAS does not typically compute time values based on the time zone of the programmer's location. One exception to this rule occurs when a SAS time (not a datetime) is computed and then formatted with either the B8601LZ*w*. format or the E8601LZ*w*. format. These two formats query the host code to determine the time zone offset. Then, the current local time and the time zone offset (based on your time zone) are displayed accordingly:

```
data _null_;
x=time();
put x=e8601lz.;
run;
```

Here is the output from the SAS log:

```
x=13:49:02-04:00
```

# Reading and Writing Datetime Values with No Time Zone Offset

In the following DATA step, SAS reads datetime values with the B8601DT*w*. and E8601DT*w*. informats and writes the datetime values by using the B8601DT*w*. and E8601DT*w*. formats:

```
data _null_;
input dtB :b8601dt15. dtE :e8601dt19.;
put dtB=b8601dt. dtE=e8601dt.;
datalines;
20180402T124022 2018-04-02T12:30:22
;
run;
```

Here is the output from the SAS log:

```
dtB=20180402T124022 dtE=2018-04-02T12:30:22
```

This example reads and writes a Java datetime value and writes the value by using the B8601DT*w.d* format:

```
data a;
input dt1 b8601dj.;
put dt1=b8601dt.;
datalines;
20180402123245
;
run;
```

Here is the output from the SAS log:

```
dt1=20180402T123245
```

# Reading and Writing Datetime Values with Time Zone Offsets

The B8601DZ*w*. and E8601DZ*w*. formats always write a datetime value for the zero meridian. The offset is always +0000 or +00:00.

In this example, SAS reads a datetime value with an offset and writes the datetime value by using an offset for the zero meridian:

```
data _null_;
x=input('2018-08-01T12:34:56-04:00',e8601dz25.);
put x=e8601dz25.;
run;
```

Here is the output from the SAS log:

```
x=2018-08-01T16:34:56+00:00
```

You use the B8601DX*w*. and E8601DX*w*. formats to adjust a datetime with a time zone offset to be the time for a specific time zone. You set the time zone by using the TIMEZONE= system option. The input value is converted to the time for the time zone and formatted using a time zone offset. This example reads the datetime value with an offset (–04:00) by using the E8601DZ*w*. informat and writes the datetime value for the time zone in Zurich (+02:00):

```
options timezone='europe/zurich';
data _null_;
x=input('2018-08-01T12:34:56-04:00',e8601dz25.);
put x=e8601dx25.;
run;
```

Here is the output from the SAS log:

```
x=2018-08-01T18:34:56+02:00
```

In this example, the TIMEZONE= system option sets the time zone ID to America/Anchorage. The datetime value is written for this time zone ID by using the

E8601DX*w*. format. The time zone offset is the difference between the America/Anchorage time zone and UTC, which is nine hours.

```
options timezone='america/anchorage';
data _null_;
t='01Feb2018T12:34:56'dt;
put t=e8601dx.;
run;
```

Here is the output from the SAS log:

```
2018-02-01T03:34:56-09:00
```

# Reading and Writing Time and Datetime Values with Time Zone Offsets for Local Times

Because time values are scalar, SAS does not typically compute time values based on the time zone of the programmer's location. One exception to this rule occurs when a SAS time (not a datetime) is computed and then formatted with either the B8601LZ*w*. format or the E8601LZ*w*. format. These two formats query the host code to determine the offset. Then, the current local time and the offset (based on your time zone) are displayed accordingly:

```
data _null_;
x=time();
put x=e8601lz.;
run;
```

Here is the output from the SAS log:

```
x=13:49:02-04:00
```

You can write a user's local datetime value by using the time zone offset. Specify the user's time zone by using the TIMEZONE= system option and the B8601LX*w*. or E8601LX*w*. format. This example writes the datetime for a user by using the time zone name PST (Pacific Time):

```
options timezone=pst;
data _null_;
x='01Feb2018T12:34:56'dt;
put x=e8601lx.;
run;
```

Here is the output from the SAS log:

```
2018-02-01T12:34:56-07:00
```

# Writing ISO 8601 Duration, Datetime, and Interval Values

## Duration, Datetime, and Interval Formats

A *duration* is the period of time that is the difference between two time points. Each time point begins with P and is followed by the date and time in either basic or extended notation. Durations can be negative or positive values and can be expressed in these forms:

- P*yyyymmdd*T*hhmmss.*
- P*yyyy-mm-dd*T*hh:mm:ss.*
- P*yyyy-mm-dd* is a span of years, months, and days.
- P*n*Y*n*M*n*DT*n*H*n*M*n*S is the number of years (Y), months (M), days (D), hours (H), minutes (M), and seconds (S).
- P*n*W specifies the number of weeks.

The *y*, *m*, and *d* placeholders must have a value, even if the value is 0.

The *n* placeholder can be 0 or a positive number. The component that contains an *n* can be omitted. For example, P0Y0M3DT0H0M0S can be written as P3D.

When you use the P*n*W notation, W must be the only component in the duration.

An interval comprises two values that represent the beginning and end of an event, and it is a duration that is anchored to a specific point in time. Intervals are represented in the following forms:

- *datetime/datetime*
- *datetime/duration*
- *duration/datetime*

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

| Time Component | ISO 8601 Notation | Example | Format |
|---|---|---|---|
| Duration - Basic Notation | P*yyyymmdd*T*hhmmssfff* | P20180915T155300 | $N8601BA |
| | –P*yyyymmdd* T*hhmmssfff* | –P20180915T155300 | $N8601BA |
| Duration - Extended Notation | P*yyyy-mm-dd*T*hh:mm:ss.fff* | P2018-09-15T15:53:00 | $N8601EA |

| Time Component | ISO 8601 Notation | Example | Format |
|---|---|---|---|
| | –P*yyyy-mm-dd*T*hh:mm:ss.fff* | –P2018-09-15T15:53:00 | $N8601EA |
| Duration - Basic and Extended Notation | P*n*Y*n*M*n*DT*n*H*n*M*n*S | P2y10m14dT20h13m45s | $N8601B<br>$N8601E |
| | –P*n*Y*n*M*n*DT*n*H*n*M*n*S | –P2y10m14dT20h13m45s | $N8601B<br>$N8601E |
| | P*n*W (weeks) | P6w | $N8601B<br>$N8601E |
| Interval - Basic Notation | *yyyymmdd*T*hhmmssfff*/yyyymmddT*hhmmssfff* | 20170915T155300/20181113T000000 | $N8601BA |
| | P*n*Y*n*M*n*DT*n*H*n*M*n*S/*yyyymmdd*T*hhmmssfff* | P2y10M14dT20h13m45s/20180915T155300 | $N8601B |
| | *yyyymmdd*T*hhmmssfff*/P*n*Y*n*M*n*DT*n*H*n*M*n*S | 20180915T155300/P2y10M14dT20h13m45s | $N8601BA |
| Interval - Extended Notation | *yyyy-mm-dd*T*hh:mm:ss.fff*/*yyyy-mm-dd*T*hh:mm:ss.fff* | 2017-09-15T15:53:00/2018-11-13T00:00:00 | $N8601EA |
| | P*n*Y*n*M*n*DT*n*H*n*M*n*S/*yyyy-mm-dd*T*hh:mm:ss.fff* | P2y10M14dT20h13m45s/2018-09-15T15:53:00 | $N8601E |
| | *yyyy-mm-dd*T*hh:mm:ss.fff*/P*n*Y*n*M*n*DT*n*H*n*M*n*S | 2018-09-15T15:53:00/P2y10M14dT20h13m45s | $N8601EA |
| Datetime - Basic Notation | *yyyymmdd*T*hhmmss.fff* +\|–*hhmm* | 20180915T155300 | $N8601BA |
| | (all blank) | | $N8601B<br>$N8601BA<br>$N8601E<br>$N8601EA |
| Datetime - Extended Notation | *yyyy-mm-dd*T*hh:mm:ss.fff*+\|–*hhmm* | 2018-09-15T15:53:00<br>+04:30 | $N8601EA |
| | (all blank) | | $N8601B<br>$N8601BA |

| Time Component | ISO 8601 Notation | Example | Format |
|---|---|---|---|
| | | | $N8601E<br>$N8601EA |

## Writing Partial and Missing Components

When any component of a date or time is not provided, it is called a partial value, and the components are considered missing. You can represent a missing component in a value by using a hyphen ( - ) or an **x**. A single hyphen represents the entire value for a given component. For example, one single hyphen can replace a four-digit year. A single x represents one character for a given component. A missing two-digit month would be written as **xx**. If the time portion is omitted when a date value is specified, the T must also be omitted.

Missing 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 an **x**. For example, in P2mT4H, the year, day, minutes, and seconds are missing and have been dropped.

Do not confuse missing components with zero values. The durations P3D and P0000-00-03 are not the same because a component value of 0 is not the same as a missing component value. Change instances of 0 to x (Pxxxx-xx-03) for this value to be considered the equivalent of P3D.

Missing components are not allowed for values that contain a time zone offset. Therefore, use 00 in place of omitted components.

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

| Format | Datetime Form | Duration Form | Example |
|---|---|---|---|
| $N8601H | *yyyy-mm-dd*T*hh:mm:ss* | P*n*Y*n*M*n*DT*n*H*n*M*n*S | --09-15T15:-:53<br>P2Y2DT4H5M6S/--09-15T15:-:00 |
| $N8601EH | *yyyy-mm-dd*T*hh:mm:ss* | P*yyyy-mm-dd*T*hh:mm:ss* | P000---02T02:55:20/2018---15T-:-:45 |
| $N8601X | *yyyy-mm-dd*T*hh:mm:ss* | P*n*Y*n*M*n*DT*n*H*n*M*n*S | P2Y2DT4H5M6S/xxxx-09-15T15:xx:00 |
| $N8601EX | *yyyy-mm-dd*T*hh:mm:ss* | P*yyyy-mm-dd*T*hh:mm:ss* | P0003-xx-02T02:55:20/2018-xx-15Txx:xx:45 |

Datetime values with missing components that are formatted with either the $N8601B. format or the $N8601BA. format are formatted in the extended notation that uses the hyphen for missing components to ensure accurate data. For example, when the month is the missing component, the value 2018---15 is written, and not 2018-15.

The extended notation with hyphens is also used instead of the basic notation if a duration is formatted using the $N8601BA. format. Using the same date, P2018---15 is written, and not P2018-15.

# Writing Truncated Duration, Datetime, and Interval Values

Duration, datetime, and interval values can be truncated when one or more lower-order values are 0 or are not significant. When SAS writes a truncated value that uses 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 that uses the $N8601X. format or the $N8601EX. format, the lower-order components are written with an **x**.

These examples show truncated values:

- `p00030202T1031`
- `2016-09-15T15/2018-09-15T15:53`
- `-p0003-03-03T-:-:-`
- `P2y3m4dT5h6m`
- `2018-09-xxTxx:xx:xx`
- `2018`

# 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 Duration, Datetime, and Interval Values

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

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

# 2

# Dictionary of Formats

# Formats Documented in Other Publications

Additional formats are documented in these publications:

- “Dictionary of Formats for NLS” in *SAS National Language Support (NLS): Reference Guide*.
- “DS2 Formats” in *SAS DS2 Language Reference*
- “FedSQL Formats” in *SAS FedSQL Language Reference*

# Formats by Category

Here are the categories for formats:

| Category | Description |
|---|---|
| CAS | Instructs SAS to write character data values from character variables in CAS.<br>See CAS for a list of formats. |
| Character | Instructs SAS to write character data values from character variables.<br>See Character for a list of formats. |
| Date | Instructs SAS to write data values from variables that represent dates.<br>See Date for a list of formats. |
| Date and Time | Instructs SAS to write data values from variables that represent dates, times, and datetimes.<br>See Date and Time for a list of formats. |
| ISO 8601 | Instructs SAS to write date, time, and datetime values using the ISO 8601 standard.<br>See ISO 8601 for a list of formats. |
| Numeric | Instructs SAS to write numeric data values from numeric variables.<br>See Numeric for a list of formats. |
| Time | Instructs SAS to write data values from variables that represent times.<br>See Time for a list of formats. |

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 "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 |
|---|---|---|
| CAS | $BASE64Xw. Format (p. 104) | Converts character data into ASCII text by using Base 64 encoding. |
| | $BINARYw. Format (p. 105) | Converts character data to binary representation. |
| | $CHARw. Format (p. 106) | Writes standard character data. |
| | $CSTR Format (p. 108) | Looks for the last non-blank character of a character argument and passes a copy of the string with a null terminator after the last non-blank character. |
| | $HEXw. Format (p. 110) | Converts character data to hexadecimal representation. |
| | $OCTALw. Format (p. 130) | Converts character data to octal representation. |
| | $QUOTEw. Format (p. 131) | Writes data values that are enclosed in double quotation marks. |
| | $REVERJw. Format (p. 133) | Writes character data in reverse order and preserves blanks. |
| | $REVERSw. Format (p. 134) | Writes character data in reverse order and left-aligns |
| | $UPCASEw. Format (p. 135) | Converts character data to uppercase. |
| | $UUIDw. Format (p. 136) | Converts character data to the Universally Unique Identifier (UUID) format. |
| | BESTw. Format (p. 141) | SAS chooses the best notation. |
| | BESTDw.p Format (p. 143) | Prints numeric values, lining up decimal places for values of similar magnitude, and prints integers without decimals. |
| | COMMAw.d Format (p. 163) | Writes numeric values with a comma that separates every three digits and a period that separates the decimal fraction. |
| | COMMAXw.d Format (p. 165) | Writes numeric values with a period that separates every three digits and a comma that separates the decimal fraction. |
| | Dw.p Format (p. 166) | Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude. |
| | DATEw. Format (p. 169) | Writes date values in the form ddmmmyy, ddmmmyyyy, or dd-mmm-yyyy. |
| | DATEAMPMw.d Format (p. 170) | Writes datetime values in the form ddmmmyy:hh:mm:ss.ss with AM or PM. |
| | DATETIMEw.d Format (p. 172) | Writes datetime values in the form ddmmmyy:hh:mm:ss.ss. |
| | DAYw. Format (p. 175) | Writes date values as the day of the month. |

| Category | Language Elements | Description |
|---|---|---|
| | DDMMYYw. Format (p. 175) | 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 two or four digits. |
| | DDMMYYxw. Format (p. 177) | 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. The special character can be a blank character, colon (:), hyphen (-),no separator, period (.), or slash (/). The year can be either two or four digits. |
| | DOLLARw.d Format (p. 180) | 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. 181) | Writes numeric values with a leading dollar sign, a period that separates every three digits, and a comma that separates the decimal fraction. |
| | DOWNAMEw. Format (p. 183) | Writes date values as the name of the day of the week. |
| | DTMONYYw. Format (p. 186) | Writes the date part of a datetime value as the month and year in the form mmmyy or mmmyyyy. |
| | DTWKDATXw. Format (p. 187) | 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. 189) | Writes the date part of a datetime value as the year in the form yy or yyyy. |
| | DTYYQCw. Format (p. 190) | Writes the date part of a datetime value as the year and the quarter and separates them with a colon (:). |
| | Ew. Format (p. 192) | Writes numeric values in scientific notation. |
| | E8601DAw. Format (p. 193) | Writes date values by using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DNw. Format (p. 194) | Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DTw.d Format (p. 195) | Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.ffffff. |
| | E8601DZw. Format (p. 199) | Writes datetime values for the zero meridian Coordinated Universal Time (UTC) by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00. |
| | E8601LZw. Format (p. 202) | 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. |

| Category | Language Elements | Description |
|---|---|---|
| | E8601TMw.d Format (p. 204) | Writes time values by using the ISO 8601 extended notation hh:mm:ss.ffffff. |
| | E8601TZw.d Format (p. 208) | Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation hh:mm:ss.<fff>+\|–hh:mm. |
| | Fw.d Format (p. 215) | Rounds the value to the nearest number that fits in the output field. |
| | FLOATw.d Format (p. 217) | Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the dth power. |
| | FRACTw. Format (p. 219) | Converts numeric values to fractions. |
| | HEXw. Format (p. 220) | Converts real binary (floating-point) values to hexadecimal representation. |
| | HHMMw.d Format (p. 221) | Writes time values as hours and minutes in the form hh:mm. |
| | HOURw.d Format (p. 224) | Writes time values as hours and decimal fractions of hours. |
| | IEEEw.d Format (p. 229) | Generates an IEEE floating-point value by multiplying a number by 10 raised to the dth power. |
| | JULIANw. Format (p. 232) | Writes date values as Julian dates in the form yyddd or yyyyddd. |
| | MDYAMPMw.d Format (p. 233) | 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. 235) | 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 two or four digits. |
| | MMDDYYxw. Format (p. 237) | 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 that 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 two or four digits. |
| | MMSSw.d Format (p. 240) | Writes time values as the number of minutes and seconds since midnight. |
| | MMYYw. Format (p. 241) | Writes date values in the form mmM<yy>yy, where M is the separator and the year appears as either two or four digits. |
| | MMYYxw. Format (p. 243) | 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. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits. |

| Category | Language Elements | Description |
|---|---|---|
| | MONNAMEw. Format (p. 245) | Writes date values as the name of the month. |
| | MONTHw. Format (p. 246) | Writes date values as the month of the year. |
| | MONYYw. Format (p. 247) | Writes date values as the month and the year in the form mmmyy or mmmyyyy. |
| | NEGPARENw.d Format (p. 249) | Writes negative numeric values in parentheses. |
| | NLBESTw. Format (p. 252) | Writes the best numerical notation based on the locale. |
| | NLDATEw. Format (p. 255) | Writes a SAS date value as a date that is appropriate for the current SAS locale. |
| | NLDATELw. Format (p. 256) | Writes a SAS date value as a date in the form month, date, year that is appropriate for the current SAS locale. |
| | NLDATEMw. Format (p. 257) | Writes a SAS date value as a date in a medium-uniform pattern that is appropriate for the current SAS locale. |
| | NLDATEMDw. Format (p. 258) | Writes the SAS date value as the name of the month and the day of the month that is appropriate for the current SAS locale. |
| | NLDATEMDLw. Format (p. 259) | Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale. |
| | NLDATEMDMw. Format (p. 260) | Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale. |
| | NLDATEMDSw. Format (p. 261) | Writes a SAS date value as the month and day of the month in a short-uniform pattern that is appropriate for the current SAS locale. |
| | NLDATEMNw. Format (p. 262) | Writes a SAS date value as the name of the month that is appropriate for the current SAS locale. |
| | NLDATESw. Format (p. 264) | Writes a SAS date value as a date string that is appropriate for the current SAS locale. |
| | NLDATEWw. Format (p. 265) | Writes a SAS date value as the date and the day of the week that is appropriate for the current SAS locale. |
| | NLDATEWNw. Format (p. 267) | Writes the SAS date value as the day of the week that is appropriate for the current SAS locale. |
| | NLDATEYMw. Format (p. 268) | Writes the SAS date value as the year and the name of the month that is appropriate for the current SAS locale. |
| | NLDATEYMLw. Format (p. 270) | Writes a SAS date value as the month and year that is appropriate for the current SAS locale. |

| Category | Language Elements | Description |
|---|---|---|
| | NLDATEYMMw. Format (p. 271) | Writes a SAS date value as the month and year with abbreviations that is appropriate for the current SAS locale. |
| | NLDATEYMSw. Format (p. 272) | Writes a SAS date value as a date and year that is appropriate for the current SAS locale. |
| | NLDATEYQw. Format (p. 273) | Writes the SAS date value as the year and the quarter that is appropriate for the current SAS locale. |
| | NLDATEYQLw. Format (p. 274) | Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale. |
| | NLDATEYQMw. Format (p. 276) | Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale. |
| | NLDATEYQSw. Format (p. 277) | Writes a SAS date value as the year and the year's quarter value (1–4) with numbers and delimiters that is appropriate for the current SAS locale. |
| | NLDATEYRw. Format (p. 278) | Writes the SAS date value as the year that is appropriate for the current SAS locale. |
| | NLDATEYWw. Format (p. 279) | Writes the SAS date value as the year and the week that is appropriate for the current SAS locale. |
| | NLDATMw. Format (p. 280) | Writes a SAS datetime value as a datetime that is appropriate for the current SAS locale. |
| | NLDATMAPw. Format (p. 281) | Wries a SAS datetime value as a datetime with a.m. or p.m. that is appropriate for the current SAS locale. |
| | NLDATMDTw. Format (p. 282) | Writes the SAS datetime value as the name of the month, day of the month and year that is appropriate for the current SAS locale. |
| | NLDATMLw. Format (p. 284) | Writes a SAS datetime value as a long representation of the date that is appropriate for the current SAS locale. |
| | NLDATMMw. Format (p. 285) | Writes a SAS datetime value as a medium representation of the date that is appropriate for the current SAS locale. |
| | NLDATMMDw. Format (p. 286) | Writes the SAS datetime value as the name of the month and the day of the month that is appropriate for the current SAS locale. |
| | NLDATMMDLw. Format (p. 287) | Writes a SAS datetime value as the full-length of the month and day of the month that is appropriate for the current SAS locale. |
| | NLDATMMDMw. Format (p. 288) | Writes a SAS datetime value as the month and day of the month using abbreviations that is appropriate for the current SAS locale. |

| Category | Language Elements | Description |
|---|---|---|
| | NLDATMMDSw. Format (p. 289) | Writes a SAS datetime value as the month and day of the month using numbers and delimiters that is appropriate for the current SAS locale. |
| | NLDATMMNw. Format (p. 290) | Writes the SAS datetime value as the name of the month that is appropriate for the current SAS locale. |
| | NLDATMSw. Format (p. 292) | Writes a SAS datetime value as the short representation of the date that is appropriate for the current SAS locale. |
| | NLDATMTMw. Format (p. 293) | Writes the time portion of a SAS datetime value as a time of day that is appropriate for the current SAS locale. |
| | NLDATMTZw. Format (p. 294) | Writes the time portion of the SAS datetime value as the time of day and time zone that is appropriate for the current SAS locale. |
| | NLDATMWw. Format (p. 295) | Writes SAS datetime values as the day of the week and the datetime that is appropriate for the current SAS locale. |
| | NLDATMWNw. Format (p. 297) | Writes a SAS datetime value as the day of the week that is appropriate for the current SAS locale. |
| | NLDATMWZw. Format (p. 298) | Writes SAS datetime values as day-of-week, datetime, and time zone values that are appropriate. |
| | NLDATMYMw. Format (p. 299) | Writes the SAS datetime value as the month and year that is appropriate for the current SAS locale. |
| | NLDATMYMLw. Format (p. 300) | Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale. |
| | NLDATMYMMw. Format (p. 301) | Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale. |
| | NLDATMYMSw. Format (p. 302) | Writes a SAS datetime value as the month and year with numbers and a delimiter that is appropriate for the current SAS locale. |
| | NLDATMYQw. Format (p. 303) | Writes the SAS datetime value as the quarter and the year that is appropriate for the current SAS locale. |
| | NLDATMYQMw. Format (p. 305) | Writes a SAS datetime value as the year's quarter (1–4) and the year that is appropriate for the current SAS locale. |
| | NLDATMYQSw. Format (p. 306) | Writes a SAS datetime value as the year and the quarter (1-4) using numbers and a delimiter that is appropriate for the current SAS locale. |
| | NLDATMYRw. Format (p. 308) | Writes the SAS datetime value as the year that is appropriate for the current SAS locale. |
| | NLDATMYWw. Format (p. 309) | Writes the SAS datetime value as the week number and the year that is appropriate for the current SAS locale. |

| Category | Language Elements | Description |
|---|---|---|
| | NLDATMZw. Format (p. 310) | Writes SAS datetime values as datetime and time zone values that are appropriate for the current SAS locale. |
| | NLMNIAEDw.d Format (p. 310) | Writes the monetary format of the international expression for the United Arab Emirates. |
| | NLMNIAUDw.d Format (p. 312) | Writes the monetary format of the international expression for Australia. |
| | NLMNIBGNw.d Format (p. 313) | Writes the monetary format of the international expression for Bulgaria. |
| | NLMNIBRLw.d Format (p. 314) | Writes the monetary format of the international expression for Brazil. |
| | NLMNICADw.d Format (p. 315) | Writes the monetary format of the international expression for Canada. |
| | NLMNICHFw.d Format (p. 316) | Writes the monetary format of the international expression for Liechtenstein and Switzerland. |
| | NLMNICNYw.d Format (p. 318) | Writes the monetary format of the international expression for China. |
| | NLMNICZKw.d Format (p. 319) | Writes the monetary format of the international expression for the Czech Republic. |
| | NLMNIDKKw.d Format (p. 320) | Writes the monetary format of the international expression for Denmark, Faroe Island, and Greenland. |
| | NLMNIEEKw.d Format (p. 321) | Writes the monetary format of the international expression for Estonia. |
| | NLMNIEGPw.d Format (p. 322) | Writes the monetary format of the international expression for Egypt. |
| | NLMNIEURw.d Format (p. 324) | Writes the monetary format of the international expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain. |
| | NLMNIGBPw.d Format (p. 325) | Writes the monetary format of the international expression for the United Kingdom. |
| | NLMNIHKDw.d Format (p. 326) | Writes the monetary format of the international expression for Hong Kong. |
| | NLMNIHRKw.d Format (p. 327) | Writes the monetary format of the international expression for Croatia. |
| | NLMNIHUFw.d Format (p. 328) | Writes the monetary format of the international expression for Hungary. |

| Category | Language Elements | Description |
|---|---|---|
| | NLMNIIDRw.d Format (p. 330) | Writes the monetary format of the international expression for Indonesia. |
| | NLMNIILSw.d Format (p. 331) | Writes the monetary format of the international expression for Israel. |
| | NLMNIINRw.d Format (p. 332) | Writes the monetary format of the international expression for India. |
| | NLMNIKRWw.d Format (p. 334) | Writes the monetary format of the international expression for South Korea. |
| | NLMNILTLw.d Format (p. 335) | Writes the monetary format of the international expression for Lithuania. |
| | NLMNILVLw.d Format (p. 337) | Writes the monetary format of the international expression for Latvia. |
| | NLMNIMOPw.d Format (p. 338) | Writes the monetary format of the international expression for Macau. |
| | NLMNIMXNw.d Format (p. 339) | Writes the monetary format of the international expression for Mexico. |
| | NLMNIMYRw.d Format (p. 340) | Writes the monetary format of the international expression for Malaysia. |
| | NLMNINOKw.d Format (p. 341) | Writes the monetary format of the international expression for Norway. |
| | NLMNINZDw.d Format (p. 343) | Writes the monetary format of the international expression for New Zealand. |
| | NLMNIPLNw.d Format (p. 344) | Writes the monetary format of the international expression for Poland. |
| | NLMNIRUBw.d Format (p. 345) | Writes the monetary format of the international expression for Russia. |
| | NLMNISEKw.d Format (p. 346) | Writes the monetary format of the international expression for Sweden. |
| | NLMNISGDw.d Format (p. 347) | Writes the monetary format of the international expression for Singapore. |
| | NLMNITHBw.d Format (p. 349) | Writes the monetary format of the international expression for Thailand. |
| | NLMNITRYw.d Format (p. 350) | Writes the monetary format of the international expression for Turkey. |
| | NLMNITWDw.d Format (p. 351) | Writes the monetary format of the international expression for Taiwan. |

| Category | Language Elements | Description |
|---|---|---|
| | NLMNIUSDw.d Format (p. 352) | Writes the monetary format of the international expression for Puerto Rico and the United States. |
| | NLMNIZARw.d Format (p. 353) | Writes the monetary format of the international expression for South Africa. |
| | NLMNLAEDx.d Format (p. 354) | Writes the monetary format of the local expression for the United Arab Emirates. |
| | NLMNLAUDw.d Format (p. 356) | Writes the monetary format of the local expression for Australia. |
| | NLMNLBGNw.d Format (p. 357) | Writes the monetary format of the local expression for Bulgaria. |
| | NLMNLBRLw.d Format (p. 358) | Writes the monetary format of the local expression for Brazil. |
| | NLMNLCADw.d Format (p. 359) | Writes the monetary format of the local expression for Canada. |
| | NLMNLCHFw.d Format (p. 360) | Writes the monetary format of the local expression for Liechtenstein and Switzerland. |
| | NLMNLCNYw.d Format (p. 361) | Writes the monetary format of the local expression for China. |
| | NLMNLCZKw.d Format (p. 363) | Writes the monetary format of the local expression for the Czech Republic. |
| | NLMNLDKKw.d Format (p. 364) | Writes the monetary format of the local expression for Denmark, Faroe Island, and Greenland. |
| | NLMNLEEKw.d Format (p. 365) | Writes the monetary format of the local expression for Estonia. |
| | NLMNLEGPw.d Format (p. 366) | Writes the monetary format of the local expression for Egypt. |
| | NLMNLEURw.d Format (p. 368) | Writes the monetary format of the local expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain. |
| | NLMNLGBPw.d Format (p. 369) | Writes the monetary format of the local expression for the United Kingdom. |
| | NLMNLHKDw.d Format (p. 370) | Writes the monetary format of the local expression for Hong Kong. |
| | NLMNLHRKw.d Format (p. 371) | Writes the monetary format of the local expression for Croatia. |

| Category | Language Elements | Description |
|---|---|---|
| | NLMNLHUFw.d Format (p. 372) | Writes the monetary format of the local expression for Hungary. |
| | NLMNLIDRw.d Format (p. 374) | Writes the monetary format of the local expression for Indonesia. |
| | NLMNLILSw.d Format (p. 375) | Writes the monetary format of the local expression for Israel. |
| | NLMNLINRw.d Format (p. 376) | Writes the monetary format of the local expression for India. |
| | NLMNLJPYw.d Format (p. 377) | Writes the monetary format of the international expression for Japan. |
| | NLMNLKRWw.d Format (p. 378) | Writes the monetary format of the local expression for South Korea. |
| | NLMNLLTLw.d Format (p. 380) | Writes the monetary format of the local expression for Lithuania. |
| | NLMNLLVLw.d Format (p. 381) | Writes the monetary format of the local expression for Latvia. |
| | NLMNLMOPw.d Format (p. 382) | Writes the monetary format of the local expression for Macau. |
| | NLMNLMXNw.d Format (p. 383) | Writes the monetary format of the local expression for Mexico. |
| | NLMNLMYRw.d Format (p. 384) | Writes the monetary format of the local expression for Malaysia. |
| | NLMNLNOKw.d Format (p. 386) | Writes the monetary format of the local expression for Norway. |
| | NLMNLNZDw.d Format (p. 387) | Writes the monetary format of the local expression for New Zealand. |
| | NLMNLPLNw.d Format (p. 388) | Writes the monetary format of the local expression for Poland. |
| | NLMNLRUBw.d Format (p. 389) | Writes the monetary format of the local expression for Russia. |
| | NLMNLSEKw.d Format (p. 390) | Writes the monetary format of the local expression for Sweden. |
| | NLMNLSGDw.d Format (p. 392) | Writes the monetary format of the local expression for Singapore. |
| | NLMNLTHBw.d Format (p. 393) | Writes the monetary format of the local expression for Thailand. |

| Category | Language Elements | Description |
|---|---|---|
| | NLMNLTRYw.d Format (p. 394) | Writes the monetary format of the local expression for Turkey. |
| | NLMNLTWDw.d Format (p. 395) | Writes the monetary format of the local expression for Taiwan. |
| | NLMNLUSDw.d Format (p. 396) | Writes the monetary format of the local expression for Puerto Rico and the United States. |
| | NLMNLZARw.d Format (p. 397) | Writes the monetary format of the local expression for South Africa. |
| | NLMNYw.d Format (p. 399) | Writes the monetary format of the local expression that is appropriate for the current SAS locale. |
| | NLMNYIw.d Format (p. 400) | Writes the monetary format of the international expression that is appropriate for the current SAS locale. |
| | NLNUMw.d Format (p. 402) | Writes the numeric format of the local expression that is appropriate for the current SAS locale. |
| | NLNUMIw.d Format (p. 404) | Writes the numeric format of the international expression that is appropriate for the current SAS locale. |
| | NLPCTw.d Format (p. 405) | Writes percentage data of the local expression that is appropriate for the current SAS locale. |
| | NLPCTIw.d Format (p. 407) | Writes percentage data of the international expression that is appropriate for the current SAS locale. |
| | NLPCTNw.d Format (p. 408) | Writes percentages, using a minus sign for negative values. |
| | NLPCTPw.d Format (p. 410) | Writes locale-specific numeric values as percentages. |
| | NLPVALUEw.d Format (p. 411) | Writes p-values of the local expression that is appropriate for the current SAS locale. |
| | NLSTRMONw.d Format (p. 412) | Writes the month name that is appropriate for the current SAS locale. |
| | NLSTRQTRw.d Format (p. 414) | Writes a numeric value as the quarter-of-the-year that is appropriate for the current SAS locale. |
| | NLSTRWKw.d Format (p. 415) | Writes a numeric value as the day-of-the-week that is appropriate for the current SAS locale. |
| | NLTIMAPw. Format (p. 416) | Writes a SAS time value as a time value with a.m. or p.m. that is appropriate for the current SAS locale. |
| | NLTIMEw. Format (p. 418) | Writes a SAS time value as a time value that is appropriate for the current SAS locale. |
| | OCTALw. Format (p. 421) | Converts numeric values to octal representation. |

| Category | Language Elements | Description |
|---|---|---|
| | PERCENTw.d Format (p. 429) | Writes numeric values as percentages. |
| | PERCENTNw.d Format (p. 431) | Produces percentages, using a minus sign for negative values. |
| | QTRw. Format (p. 440) | Writes date values as the quarter of the year. |
| | QTRRw. Format (p. 441) | Writes date values as the quarter of the year in Roman numerals. |
| | ROMANw. Format (p. 444) | Writes numeric values as Roman numerals. |
| | TIMEw.d Format (p. 465) | Writes time values as hours, minutes, and seconds in the form hh:mm:ss.ss. |
| | TIMEAMPMw.d Format (p. 468) | 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. 470) | Writes SAS time values and the time portion of SAS datetime values in the form hh:mm:ss.ss. |
| | VAXRBw.d Format (p. 473) | Writes real binary (floating-point) data in VMS format. |
| | w.d Format (p. 476) | Writes standard numeric data one digit per byte. |
| | WEEKDATXw. Format (p. 480) | 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. 482) | Writes date values as the day of the week. |
| | YEARw. Format (p. 495) | Writes date values as the year. |
| | YYMMw. Format (p. 497) | 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 two or four digits. |
| | YYMMDDw. Format (p. 498) | Writes date values in the form yymmdd or <yy>yy-mm-dd, where a hyphen is the separator and the year appears as either two or four digits. |
| | YYMMDDxw. Format (p. 501) | 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 that 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 two or four digits. |
| | YYMMxw. Format (p. 503) | 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. |

| Category | Language Elements | Description |
|---|---|---|
| | YYMONw. Format (p. 505) | Writes date values in the form yymmm or yyyymmm. |
| | YYQw. Format (p. 506) | Writes date values in the form <yy>yyQq, where Q is the separator, the year appears as either two or four digits, and q is the quarter of the year. |
| | YYQxw. Format (p. 508) | 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. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits. |
| | YYQRw. Format (p. 510) | Writes date values in the form <yy>yyQqr, where Q is the separator, the year appears as either two or four digits, and qr is the quarter of the year expressed in Roman numerals. |
| | YYQRxw. Format (p. 512) | 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. The special character separates the year and the quarter or the year. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits, and qr is the quarter of the year expressed in Roman numerals. |
| | YYQZw. Format (p. 514) | Writes SAS date values in the form <yy><qq>, the year appears as 2 or 4 digits, and qq is the quarter of the year. |
| | YYWEEKUw. Format (p. 515) | Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information. |
| | YYWEEKVw. Format (p. 517) | Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information. |
| | YYWEEKWw. Format (p. 519) | Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information. |
| | Zw.d Format (p. 521) | Writes standard numeric data with leading 0s. |
| Character | $ASCIIw. Format (p. 102) | Converts native format character data to ASCII representation. |
| | $BASE64Xw. Format (p. 104) | Converts character data into ASCII text by using Base 64 encoding. |
| | $BINARYw. Format (p. 105) | Converts character data to binary representation. |
| | $CHARw. Format (p. 106) | Writes standard character data. |
| | $CSTR Format (p. 108) | Looks for the last non-blank character of a character argument and passes a copy of the string with a null terminator after the last non-blank character. |
| | $EBCDICw. Format (p. 109) | Converts native format character data to EBCDIC representation. |

| Category | Language Elements | Description |
|---|---|---|
| | $HEXw. Format (p. 110) | Converts character data to hexadecimal representation. |
| | $MSGCASEw. Format (p. 112) | Writes character data in uppercase when the MSGCASE system option is in effect. |
| | $OCTALw. Format (p. 130) | Converts character data to octal representation. |
| | $QUOTEw. Format (p. 131) | Writes data values that are enclosed in double quotation marks. |
| | $REVERJw. Format (p. 133) | Writes character data in reverse order and preserves blanks. |
| | $REVERSw. Format (p. 134) | Writes character data in reverse order and left-aligns |
| | $UPCASEw. Format (p. 135) | Converts character data to uppercase. |
| | $UUIDw. Format (p. 136) | Converts character data to the Universally Unique Identifier (UUID) format. |
| | $VARYINGw. Format (p. 137) | Writes character data of varying length. |
| | $w. Format (p. 140) | Writes standard character data. |
| Date | B8601DAw. Format (p. 146) | Writes date values by using the ISO 8601 basic notation yyyymmdd. |
| | B8601DNw. Format (p. 148) | Writes dates from datetime values by using the ISO 8601 basic notation yyyymmdd. |
| | DATEw. Format (p. 169) | Writes date values in the form ddmmmyy, ddmmmyyyy, or dd-mmm-yyyy. |
| | DAYw. Format (p. 175) | Writes date values as the day of the month. |
| | DDMMYYw. Format (p. 175) | 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 two or four digits. |
| | DDMMYYxw. Format (p. 177) | 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. The special character can be a blank character, colon (:), hyphen (-),no separator, period (.), or slash (/). The year can be either two or four digits. |
| | DOWNAMEw. Format (p. 183) | Writes date values as the name of the day of the week. |
| | DTDATEw. Format (p. 184) | Expects a datetime value as input and writes date values in the form ddmmmyy or ddmmmyyyy. |

| Category | Language Elements | Description |
|---|---|---|
| | DTMONYYw. Format (p. 186) | Writes the date part of a datetime value as the month and year in the form mmmyy or mmmyyyy. |
| | DTWKDATXw. Format (p. 187) | 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. 189) | Writes the date part of a datetime value as the year in the form yy or yyyy. |
| | DTYYQCw. Format (p. 190) | Writes the date part of a datetime value as the year and the quarter and separates them with a colon (:). |
| | E8601DAw. Format (p. 193) | Writes date values by using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DNw. Format (p. 194) | Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd. |
| | JULDAYw. Format (p. 230) | Writes date values as the Julian day of the year. |
| | JULIANw. Format (p. 232) | Writes date values as Julian dates in the form yyddd or yyyyddd. |
| | MMDDYYw. Format (p. 235) | 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 two or four digits. |
| | MMDDYYxw. Format (p. 237) | 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 that 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 two or four digits. |
| | MMYYw. Format (p. 241) | Writes date values in the form mmM<yy>yy, where M is the separator and the year appears as either two or four digits. |
| | MMYYxw. Format (p. 243) | 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. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits. |
| | MONNAMEw. Format (p. 245) | Writes date values as the name of the month. |
| | MONTHw. Format (p. 246) | Writes date values as the month of the year. |
| | MONYYw. Format (p. 247) | Writes date values as the month and the year in the form mmmyy or mmmyyyy. |
| | NENGOw. Format (p. 251) | Writes date values as Japanese dates in the form e.yymmdd. |

| Category | Language Elements | Description |
|---|---|---|
| | NLDATEw. Format (p. 255) | Writes a SAS date value as a date that is appropriate for the current SAS locale. |
| | NLDATELw. Format (p. 256) | Writes a SAS date value as a date in the form month, date, year that is appropriate for the current SAS locale. |
| | NLDATEMw. Format (p. 257) | Writes a SAS date value as a date in a medium-uniform pattern that is appropriate for the current SAS locale. |
| | NLDATEMDw. Format (p. 258) | Writes the SAS date value as the name of the month and the day of the month that is appropriate for the current SAS locale. |
| | NLDATEMDLw. Format (p. 259) | Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale. |
| | NLDATEMDMw. Format (p. 260) | Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale. |
| | NLDATEMDSw. Format (p. 261) | Writes a SAS date value as the month and day of the month in a short-uniform pattern that is appropriate for the current SAS locale. |
| | NLDATEMNw. Format (p. 262) | Writes a SAS date value as the name of the month that is appropriate for the current SAS locale. |
| | NLDATESw. Format (p. 264) | Writes a SAS date value as a date string that is appropriate for the current SAS locale. |
| | NLDATEWw. Format (p. 265) | Writes a SAS date value as the date and the day of the week that is appropriate for the current SAS locale. |
| | NLDATEWNw. Format (p. 267) | Writes the SAS date value as the day of the week that is appropriate for the current SAS locale. |
| | NLDATEYMw. Format (p. 268) | Writes the SAS date value as the year and the name of the month that is appropriate for the current SAS locale. |
| | NLDATEYMLw. Format (p. 270) | Writes a SAS date value as the month and year that is appropriate for the current SAS locale. |
| | NLDATEYMMw. Format (p. 271) | Writes a SAS date value as the month and year with abbreviations that is appropriate for the current SAS locale. |
| | NLDATEYMSw. Format (p. 272) | Writes a SAS date value as a date and year that is appropriate for the current SAS locale. |
| | NLDATEYQw. Format (p. 273) | Writes the SAS date value as the year and the quarter that is appropriate for the current SAS locale. |
| | NLDATEYQLw. Format (p. 274) | Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale. |

| Category | Language Elements | Description |
|---|---|---|
| | NLDATEYQMw. Format (p. 276) | Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale. |
| | NLDATEYQSw. Format (p. 277) | Writes a SAS date value as the year and the year's quarter value (1–4) with numbers and delimiters that is appropriate for the current SAS locale. |
| | NLDATEYRw. Format (p. 278) | Writes the SAS date value as the year that is appropriate for the current SAS locale. |
| | NLDATEYWw. Format (p. 279) | Writes the SAS date value as the year and the week that is appropriate for the current SAS locale. |
| | NLDATMDTw. Format (p. 282) | Writes the SAS datetime value as the name of the month, day of the month and year that is appropriate for the current SAS locale. |
| | NLDATMMDw. Format (p. 286) | Writes the SAS datetime value as the name of the month and the day of the month that is appropriate for the current SAS locale. |
| | NLDATMMDLw. Format (p. 287) | Writes a SAS datetime value as the full-length of the month and day of the month that is appropriate for the current SAS locale. |
| | NLDATMMDMw. Format (p. 288) | Writes a SAS datetime value as the month and day of the month using abbreviations that is appropriate for the current SAS locale. |
| | NLDATMMDSw. Format (p. 289) | Writes a SAS datetime value as the month and day of the month using numbers and delimiters that is appropriate for the current SAS locale. |
| | NLDATMMNw. Format (p. 290) | Writes the SAS datetime value as the name of the month that is appropriate for the current SAS locale. |
| | NLDATMWZw. Format (p. 298) | Writes SAS datetime values as day-of-week, datetime, and time zone values that are appropriate. |
| | NLDATMYMLw. Format (p. 300) | Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale. |
| | NLDATMYMMw. Format (p. 301) | Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale. |
| | NLDATMYMSw. Format (p. 302) | Writes a SAS datetime value as the month and year with numbers and a delimiter that is appropriate for the current SAS locale. |
| | NLDATMYQLw. Format (p. 304) | Writes a SAS datetime value as the year's quarter value (1–4) and the year that is appropriate for the current SAS locale. |

| Category | Language Elements | Description |
|---|---|---|
| | NLDATMYQMw. Format (p. 305) | Writes a SAS datetime value as the year's quarter (1–4) and the year that is appropriate for the current SAS locale. |
| | NLDATMYQSw. Format (p. 306) | Writes a SAS datetime value as the year and the quarter (1-4) using numbers and a delimiter that is appropriate for the current SAS locale. |
| | PDJULGw. Format (p. 426) | Writes packed Julian date values in the hexadecimal format yyyydddF for IBM. |
| | PDJULIw. Format (p. 427) | Writes packed Julian date values in the hexadecimal format ccyydddF for IBM. |
| | QTRw. Format (p. 440) | Writes date values as the quarter of the year. |
| | QTRRw. Format (p. 441) | Writes date values as the quarter of the year in Roman numerals. |
| | WEEKDATEw. Format (p. 478) | 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. 480) | 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. 482) | Writes date values as the day of the week. |
| | WEEKUw. Format (p. 483) | Writes a week number in decimal format by using the U algorithm. |
| | WEEKVw. Format (p. 485) | Writes a week number in decimal format by using the V algorithm. |
| | WEEKWw. Format (p. 487) | Writes a week number in decimal format by using the W algorithm. |
| | WORDDATEw. Format (p. 489) | Writes date values as the name of the month, the day, and the year in the form month-name dd, yyyy. |
| | WORDDATXw. Format (p. 491) | Writes date values as the day, the name of the month, and the year in the form dd month-name yyyy. |
| | YEARw. Format (p. 495) | Writes date values as the year. |
| | YYMMw. Format (p. 497) | 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 two or four digits. |
| | YYMMDDw. Format (p. 498) | Writes date values in the form yymmdd or <yy>yy-mm-dd, where a hyphen is the separator and the year appears as either two or four digits. |

| Category | Language Elements | Description |
|---|---|---|
| | YYMMDDxw. Format (p. 501) | 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 that 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 two or four digits. |
| | YYMMxw. Format (p. 503) | 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. |
| | YYMONw. Format (p. 505) | Writes date values in the form yymmm or yyyymmm. |
| | YYQw. Format (p. 506) | Writes date values in the form <yy>yyQq, where Q is the separator, the year appears as either two or four digits, and q is the quarter of the year. |
| | YYQxw. Format (p. 508) | 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. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits. |
| | YYQRw. Format (p. 510) | Writes date values in the form <yy>yyQqr, where Q is the separator, the year appears as either two or four digits, and qr is the quarter of the year expressed in Roman numerals. |
| | YYQRxw. Format (p. 512) | 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. The special character separates the year and the quarter or the year. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits, and qr is the quarter of the year expressed in Roman numerals. |
| Date and Time | $N8601Bw.d Format (p. 113) | Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyymmddThhmmss. |
| | $N8601BAw.d Format (p. 115) | Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PyyyymmddThhmmss and yyyymmddThhmmss. |
| | $N8601Ew.d Format (p. 117) | Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss. |
| | $N8601EAw.d Format (p. 120) | 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. 122) | Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and |

| Category | Language Elements | Description |
|---|---|---|
| | | yyyy-mm-ddThh:mm:ss, using a hyphen (-) for omitted components. |
| | $N8601EXw.d Format (p. 124) | 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. 126) | 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. 128) | 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. |
| | B8601DNw. Format (p. 148) | Writes dates from datetime values by using the ISO 8601 basic notation yyyymmdd. |
| | B8601DTw.d Format (p. 149) | Writes datetime values by using the ISO 8601 basic notation yyyymmddThhmmss<ffffff>. |
| | B8601DXw. Format (p. 151) | Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+hhmm. |
| | B8601DZw. Format (p. 153) | 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. |
| | B8601LXw. Format (p. 154) | Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation yyyymmddThhmmss+\|–hhmm. |
| | B8601LZw. Format (p. 156) | 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. |
| | DATEAMPMw.d Format (p. 170) | Writes datetime values in the form ddmmmyy:hh:mm:ss.ss with AM or PM. |
| | DATETIMEw.d Format (p. 172) | Writes datetime values in the form ddmmmyy:hh:mm:ss.ss. |
| | DTDATEw. Format (p. 184) | Expects a datetime value as input and writes date values in the form ddmmmyy or ddmmmyyyy. |
| | DTMONYYw. Format (p. 186) | Writes the date part of a datetime value as the month and year in the form mmmyy or mmmyyyy. |

| Category | Language Elements | Description |
|---|---|---|
| | DTWKDATXw. Format (p. 187) | 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. 189) | Writes the date part of a datetime value as the year in the form yy or yyyy. |
| | DTYYQCw. Format (p. 190) | Writes the date part of a datetime value as the year and the quarter and separates them with a colon (:). |
| | E8601DNw. Format (p. 194) | Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DTw.d Format (p. 195) | Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.ffffff. |
| | E8601DXw. Format (p. 197) | Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+hh:mm. |
| | E8601DZw. Format (p. 199) | Writes datetime values for the zero meridian Coordinated Universal Time (UTC) by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00. |
| | E8601LXw. Format (p. 201) | Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss+\|–hh:mm. |
| | MDYAMPMw.d Format (p. 233) | Writes datetime values in the form mm/dd/yy<yy> hh:mm AM\|PM. The year can be either two or four digits. |
| | NLDATMw. Format (p. 280) | Writes a SAS datetime value as a datetime that is appropriate for the current SAS locale. |
| | NLDATMAPw. Format (p. 281) | Wries a SAS datetime value as a datetime with a.m. or p.m. that is appropriate for the current SAS locale. |
| | NLDATMDTw. Format (p. 282) | Writes the SAS datetime value as the name of the month, day of the month and year that is appropriate for the current SAS locale. |
| | NLDATMLw. Format (p. 284) | Writes a SAS datetime value as a long representation of the date that is appropriate for the current SAS locale. |
| | NLDATMMw. Format (p. 285) | Writes a SAS datetime value as a medium representation of the date that is appropriate for the current SAS locale. |
| | NLDATMMDw. Format (p. 286) | Writes the SAS datetime value as the name of the month and the day of the month that is appropriate for the current SAS locale. |
| | NLDATMMNw. Format (p. 290) | Writes the SAS datetime value as the name of the month that is appropriate for the current SAS locale. |

| Category | Language Elements | Description |
|---|---|---|
| | NLDATMSw. Format (p. 292) | Writes a SAS datetime value as the short representation of the date that is appropriate for the current SAS locale. |
| | NLDATMTMw. Format (p. 293) | Writes the time portion of a SAS datetime value as a time of day that is appropriate for the current SAS locale. |
| | NLDATMTZw. Format (p. 294) | Writes the time portion of the SAS datetime value as the time of day and time zone that is appropriate for the current SAS locale. |
| | NLDATMWw. Format (p. 295) | Writes SAS datetime values as the day of the week and the datetime that is appropriate for the current SAS locale. |
| | NLDATMWNw. Format (p. 297) | Writes a SAS datetime value as the day of the week that is appropriate for the current SAS locale. |
| | NLDATMWZw. Format (p. 298) | Writes SAS datetime values as day-of-week, datetime, and time zone values that are appropriate. |
| | NLDATMYMw. Format (p. 299) | Writes the SAS datetime value as the month and year that is appropriate for the current SAS locale. |
| | NLDATMYQw. Format (p. 303) | Writes the SAS datetime value as the quarter and the year that is appropriate for the current SAS locale. |
| | NLDATMYRw. Format (p. 308) | Writes the SAS datetime value as the year that is appropriate for the current SAS locale. |
| | NLDATMYWw. Format (p. 309) | Writes the SAS datetime value as the week number and the year that is appropriate for the current SAS locale. |
| | NLDATMZw. Format (p. 310) | Writes SAS datetime values as datetime and time zone values that are appropriate for the current SAS locale. |
| | NLTIMAPw. Format (p. 416) | Writes a SAS time value as a time value with a.m. or p.m. that is appropriate for the current SAS locale. |
| | NLTIMEw. Format (p. 418) | Writes a SAS time value as a time value that is appropriate for the current SAS locale. |
| | TIMEAMPMw.d Format (p. 468) | 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. 470) | Writes SAS time values and the time portion of SAS datetime values in the form hh:mm:ss.ss. |
| | YYQZw. Format (p. 514) | Writes SAS date values in the form <yy><qq>, the year appears as 2 or 4 digits, and qq is the quarter of the year. |
| | YYWEEKUw. Format (p. 515) | Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information. |
| | YYWEEKVw. Format (p. 517) | Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information. |

| Category | Language Elements | Description |
|---|---|---|
| | YYWEEKWw. Format (p. 519) | Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information. |
| ISO 8601 | $N8601Bw.d Format (p. 113) | Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyymmddThhmmss. |
| | $N8601BAw.d Format (p. 115) | Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PyyyymmddThhmmss and yyyymmddThhmmss. |
| | $N8601Ew.d Format (p. 117) | Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss. |
| | $N8601EAw.d Format (p. 120) | 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. 122) | 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. 124) | 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. 126) | 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. 128) | 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. 146) | Writes date values by using the ISO 8601 basic notation yyyymmdd. |
| | B8601DNw. Format (p. 148) | Writes dates from datetime values by using the ISO 8601 basic notation yyyymmdd. |
| | B8601DTw.d Format (p. 149) | Writes datetime values by using the ISO 8601 basic notation yyyymmddThhmmss<ffffff>. |
| | B8601DXw. Format (p. 151) | Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+hhmm. |

| Category | Language Elements | Description |
|---|---|---|
| | B8601DZw. Format (p. 153) | 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. |
| | B8601LXw. Format (p. 154) | Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation yyyymmddThhmmss+\|–hhmm. |
| | B8601LZw. Format (p. 156) | 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. 158) | Writes time values by using the ISO 8601 basic notation hhmmss<ffff>. |
| | B8601TXw. Format (p. 159) | Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss+\|-hhmm. |
| | B8601TZw. Format (p. 161) | 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. 193) | Writes date values by using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DNw. Format (p. 194) | Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DTw.d Format (p. 195) | Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.ffffff. |
| | E8601DXw. Format (p. 197) | Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+hh:mm. |
| | E8601DZw. Format (p. 199) | Writes datetime values for the zero meridian Coordinated Universal Time (UTC) by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00. |
| | E8601LXw. Format (p. 201) | Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss+\|–hh:mm. |
| | E8601LZw. Format (p. 202) | 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. 204) | Writes time values by using the ISO 8601 extended notation hh:mm:ss.ffffff. |

| Category | Language Elements | Description |
|---|---|---|
| | E8601TXw. Format (p. 206) | Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 extended time notation hh:mm:ss+\|–hh:mm. |
| | E8601TZw.d Format (p. 208) | Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation hh:mm:ss.<fff>+\|–hh:mm. |
| Numeric | BESTw. Format (p. 141) | SAS chooses the best notation. |
| | BESTDw.p Format (p. 143) | Prints numeric values, lining up decimal places for values of similar magnitude, and prints integers without decimals. |
| | BINARYw. Format (p. 145) | Converts numeric values to binary representation. |
| | COMMAw.d Format (p. 163) | Writes numeric values with a comma that separates every three digits and a period that separates the decimal fraction. |
| | COMMAXw.d Format (p. 165) | Writes numeric values with a period that separates every three digits and a comma that separates the decimal fraction. |
| | Dw.p Format (p. 166) | Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude. |
| | DOLLARw.d Format (p. 180) | 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. 181) | 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. 192) | Writes numeric values in scientific notation. |
| | EUROw.d Format (p. 210) | Writes numeric values with a leading euro symbol (E), a comma that separates every three digits, and a period that separates the decimal fraction. |
| | EUROXw.d Format (p. 213) | Writes numeric values with a leading euro symbol (E), a period that separates every three digits, and a comma that separates the decimal fraction. |
| | Fw.d Format (p. 215) | Rounds the value to the nearest number that fits in the output field. |
| | FLOATw.d Format (p. 217) | Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the dth power. |
| | FRACTw. Format (p. 219) | Converts numeric values to fractions. |
| | HEXw. Format (p. 220) | Converts real binary (floating-point) values to hexadecimal representation. |

| Category | Language Elements | Description |
|---|---|---|
| | IBw.d Format (p. 225) | Writes native integer binary (fixed-point) values, including negative values. |
| | IBRw.d Format (p. 227) | Writes integer binary (fixed-point) values in Intel and DEC formats. |
| | IEEEw.d Format (p. 229) | Generates an IEEE floating-point value by multiplying a number by 10 raised to the dth power. |
| | NEGPARENw.d Format (p. 249) | Writes negative numeric values in parentheses. |
| | NLBESTw. Format (p. 252) | Writes the best numerical notation based on the locale. |
| | NLMNIAEDw.d Format (p. 310) | Writes the monetary format of the international expression for the United Arab Emirates. |
| | NLMNIAUDw.d Format (p. 312) | Writes the monetary format of the international expression for Australia. |
| | NLMNIBGNw.d Format (p. 313) | Writes the monetary format of the international expression for Bulgaria. |
| | NLMNIBRLw.d Format (p. 314) | Writes the monetary format of the international expression for Brazil. |
| | NLMNICADw.d Format (p. 315) | Writes the monetary format of the international expression for Canada. |
| | NLMNICHFw.d Format (p. 316) | Writes the monetary format of the international expression for Liechtenstein and Switzerland. |
| | NLMNICNYw.d Format (p. 318) | Writes the monetary format of the international expression for China. |
| | NLMNICZKw.d Format (p. 319) | Writes the monetary format of the international expression for the Czech Republic. |
| | NLMNIDKKw.d Format (p. 320) | Writes the monetary format of the international expression for Denmark, Faroe Island, and Greenland. |
| | NLMNIEEKw.d Format (p. 321) | Writes the monetary format of the international expression for Estonia. |
| | NLMNIEGPw.d Format (p. 322) | Writes the monetary format of the international expression for Egypt. |
| | NLMNIEURw.d Format (p. 324) | Writes the monetary format of the international expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain. |

| Category | Language Elements | Description |
|---|---|---|
| | NLMNIGBPw.d Format (p. 325) | Writes the monetary format of the international expression for the United Kingdom. |
| | NLMNIHKDw.d Format (p. 326) | Writes the monetary format of the international expression for Hong Kong. |
| | NLMNIHRKw.d Format (p. 327) | Writes the monetary format of the international expression for Croatia. |
| | NLMNIHUFw.d Format (p. 328) | Writes the monetary format of the international expression for Hungary. |
| | NLMNIIDRw.d Format (p. 330) | Writes the monetary format of the international expression for Indonesia. |
| | NLMNIILSw.d Format (p. 331) | Writes the monetary format of the international expression for Israel. |
| | NLMNIINRw.d Format (p. 332) | Writes the monetary format of the international expression for India. |
| | NLMNIJPYw.d Format (p. 333) | Writes the monetary format of the international expression for Japan. |
| | NLMNIKRWw.d Format (p. 334) | Writes the monetary format of the international expression for South Korea. |
| | NLMNILTLw.d Format (p. 335) | Writes the monetary format of the international expression for Lithuania. |
| | NLMNILVLw.d Format (p. 337) | Writes the monetary format of the international expression for Latvia. |
| | NLMNIMOPw.d Format (p. 338) | Writes the monetary format of the international expression for Macau. |
| | NLMNIMXNw.d Format (p. 339) | Writes the monetary format of the international expression for Mexico. |
| | NLMNIMYRw.d Format (p. 340) | Writes the monetary format of the international expression for Malaysia. |
| | NLMNINOKw.d Format (p. 341) | Writes the monetary format of the international expression for Norway. |
| | NLMNINZDw.d Format (p. 343) | Writes the monetary format of the international expression for New Zealand. |
| | NLMNIPLNw.d Format (p. 344) | Writes the monetary format of the international expression for Poland. |
| | NLMNIRUBw.d Format (p. 345) | Writes the monetary format of the international expression for Russia. |

| Category | Language Elements | Description |
|---|---|---|
| | NLMNISEKw.d Format (p. 346) | Writes the monetary format of the international expression for Sweden. |
| | NLMNISGDw.d Format (p. 347) | Writes the monetary format of the international expression for Singapore. |
| | NLMNITHBw.d Format (p. 349) | Writes the monetary format of the international expression for Thailand. |
| | NLMNITRYw.d Format (p. 350) | Writes the monetary format of the international expression for Turkey. |
| | NLMNITWDw.d Format (p. 351) | Writes the monetary format of the international expression for Taiwan. |
| | NLMNIUSDw.d Format (p. 352) | Writes the monetary format of the international expression for Puerto Rico and the United States. |
| | NLMNIZARw.d Format (p. 353) | Writes the monetary format of the international expression for South Africa. |
| | NLMNLAEDx.d Format (p. 354) | Writes the monetary format of the local expression for the United Arab Emirates. |
| | NLMNLAUDw.d Format (p. 356) | Writes the monetary format of the local expression for Australia. |
| | NLMNLBGNw.d Format (p. 357) | Writes the monetary format of the local expression for Bulgaria. |
| | NLMNLBRLw.d Format (p. 358) | Writes the monetary format of the local expression for Brazil. |
| | NLMNLCADw.d Format (p. 359) | Writes the monetary format of the local expression for Canada. |
| | NLMNLCHFw.d Format (p. 360) | Writes the monetary format of the local expression for Liechtenstein and Switzerland. |
| | NLMNLCNYw.d Format (p. 361) | Writes the monetary format of the local expression for China. |
| | NLMNLCZKw.d Format (p. 363) | Writes the monetary format of the local expression for the Czech Republic. |
| | NLMNLDKKw.d Format (p. 364) | Writes the monetary format of the local expression for Denmark, Faroe Island, and Greenland. |
| | NLMNLEEKw.d Format (p. 365) | Writes the monetary format of the local expression for Estonia. |
| | NLMNLEGPw.d Format (p. 366) | Writes the monetary format of the local expression for Egypt. |

| Category | Language Elements | Description |
|---|---|---|
| | NLMNLEURw.d Format (p. 368) | Writes the monetary format of the local expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain. |
| | NLMNLGBPw.d Format (p. 369) | Writes the monetary format of the local expression for the United Kingdom. |
| | NLMNLHKDw.d Format (p. 370) | Writes the monetary format of the local expression for Hong Kong. |
| | NLMNLHRKw.d Format (p. 371) | Writes the monetary format of the local expression for Croatia. |
| | NLMNLHUFw.d Format (p. 372) | Writes the monetary format of the local expression for Hungary. |
| | NLMNLIDRw.d Format (p. 374) | Writes the monetary format of the local expression for Indonesia. |
| | NLMNLILSw.d Format (p. 375) | Writes the monetary format of the local expression for Israel. |
| | NLMNLINRw.d Format (p. 376) | Writes the monetary format of the local expression for India. |
| | NLMNLJPYw.d Format (p. 377) | Writes the monetary format of the international expression for Japan. |
| | NLMNLKRWw.d Format (p. 378) | Writes the monetary format of the local expression for South Korea. |
| | NLMNLLTLw.d Format (p. 380) | Writes the monetary format of the local expression for Lithuania. |
| | NLMNLLVLw.d Format (p. 381) | Writes the monetary format of the local expression for Latvia. |
| | NLMNLMOPw.d Format (p. 382) | Writes the monetary format of the local expression for Macau. |
| | NLMNLMXNw.d Format (p. 383) | Writes the monetary format of the local expression for Mexico. |
| | NLMNLMYRw.d Format (p. 384) | Writes the monetary format of the local expression for Malaysia. |
| | NLMNLNOKw.d Format (p. 386) | Writes the monetary format of the local expression for Norway. |
| | NLMNLNZDw.d Format (p. 387) | Writes the monetary format of the local expression for New Zealand. |

| Category | Language Elements | Description |
|---|---|---|
| | NLMNLPLNw.d Format (p. 388) | Writes the monetary format of the local expression for Poland. |
| | NLMNLRUBw.d Format (p. 389) | Writes the monetary format of the local expression for Russia. |
| | NLMNLSEKw.d Format (p. 390) | Writes the monetary format of the local expression for Sweden. |
| | NLMNLSGDw.d Format (p. 392) | Writes the monetary format of the local expression for Singapore. |
| | NLMNLTHBw.d Format (p. 393) | Writes the monetary format of the local expression for Thailand. |
| | NLMNLTRYw.d Format (p. 394) | Writes the monetary format of the local expression for Turkey. |
| | NLMNLTWDw.d Format (p. 395) | Writes the monetary format of the local expression for Taiwan. |
| | NLMNLUSDw.d Format (p. 396) | Writes the monetary format of the local expression for Puerto Rico and the United States. |
| | NLMNLZARw.d Format (p. 397) | Writes the monetary format of the local expression for South Africa. |
| | NLMNYw.d Format (p. 399) | Writes the monetary format of the local expression that is appropriate for the current SAS locale. |
| | NLMNYIw.d Format (p. 400) | Writes the monetary format of the international expression that is appropriate for the current SAS locale. |
| | NLNUMw.d Format (p. 402) | Writes the numeric format of the local expression that is appropriate for the current SAS locale. |
| | NLNUMIw.d Format (p. 404) | Writes the numeric format of the international expression that is appropriate for the current SAS locale. |
| | NLPCTw.d Format (p. 405) | Writes percentage data of the local expression that is appropriate for the current SAS locale. |
| | NLPCTIw.d Format (p. 407) | Writes percentage data of the international expression that is appropriate for the current SAS locale. |
| | NLPCTNw.d Format (p. 408) | Writes percentages, using a minus sign for negative values. |
| | NLPCTPw.d Format (p. 410) | Writes locale-specific numeric values as percentages. |
| | NLPVALUEw.d Format (p. 411) | Writes p-values of the local expression that is appropriate for the current SAS locale. |
| | NLSTRMONw.d Format (p. 412) | Writes the month name that is appropriate for the current SAS locale. |

| Category | Language Elements | Description |
|---|---|---|
| | NLSTRQTRw.d Format (p. 414) | Writes a numeric value as the quarter-of-the-year that is appropriate for the current SAS locale. |
| | NLSTRWKw.d Format (p. 415) | Writes a numeric value as the day-of-the-week that is appropriate for the current SAS locale. |
| | NUMXw.d Format (p. 419) | Writes numeric values with a comma in place of the decimal point. |
| | OCTALw. Format (p. 421) | Converts numeric values to octal representation. |
| | ODDSRw.d Format (p. 422) | Writes odds ratios. |
| | PDw.d Format (p. 424) | Writes data in packed decimal format. |
| | PERCENTw.d Format (p. 429) | Writes numeric values as percentages. |
| | PERCENTNw.d Format (p. 431) | Produces percentages, using a minus sign for negative values. |
| | PIBw.d Format (p. 433) | Writes positive integer binary (fixed-point) values. |
| | PIBRw.d Format (p. 435) | Writes positive integer binary (fixed-point) values in Intel and DEC formats. |
| | PKw.d Format (p. 437) | Writes data in unsigned packed decimal format. |
| | PVALUEw.d Format (p. 438) | Writes p-values. |
| | RBw.d Format (p. 442) | Writes real binary data (floating-point) in real binary format. |
| | ROMANw. Format (p. 444) | Writes numeric values as Roman numerals. |
| | S370FFw.d Format (p. 445) | Writes native standard numeric data in IBM mainframe format. |
| | S370FIBw.d Format (p. 446) | Writes integer binary (fixed-point) values, including negative values, in IBM mainframe format. |
| | S370FIBUw.d Format (p. 448) | Writes unsigned integer binary (fixed-point) values in IBM mainframe format. |
| | S370FPDw.d Format (p. 450) | Writes packed decimal data in IBM mainframe format. |
| | S370FPDUw.d Format (p. 452) | Writes unsigned packed decimal data in IBM mainframe format. |
| | S370FPIBw.d Format (p. 453) | Writes positive integer binary (fixed-point) values in IBM mainframe format. |
| | S370FRBw.d Format (p. 455) | Writes real binary (floating-point) data in IBM mainframe format. |

| Category | Language Elements | Description |
|---|---|---|
| | S370FZDw.d Format (p. 457) | Writes zoned decimal data in IBM mainframe format. |
| | S370FZDLw.d Format (p. 458) | Writes zoned decimal leading-sign data in IBM mainframe format. |
| | S370FZDSw.d Format (p. 460) | Writes zoned decimal separate leading-sign data in IBM mainframe format. |
| | S370FZDTw.d Format (p. 461) | Writes zoned decimal separate trailing-sign data in IBM mainframe format. |
| | S370FZDUw.d Format (p. 463) | Writes unsigned zoned decimal data in IBM mainframe format. |
| | SSNw. Format (p. 464) | Writes Social Security numbers. |
| | VAXRBw.d Format (p. 473) | Writes real binary (floating-point) data in VMS format. |
| | VMSZNw.d Format (p. 474) | Generates VMS and MicroFocus COBOL zoned numeric data. |
| | w.d Format (p. 476) | Writes standard numeric data one digit per byte. |
| | WORDFw. Format (p. 493) | Writes numeric values as words with fractions that are shown numerically. |
| | WORDSw. Format (p. 494) | Writes numeric values as words. |
| | YYMONw. Format (p. 505) | Writes date values in the form yymmm or yyyymmm. |
| | YYQRxw. Format (p. 512) | 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. The special character separates the year and the quarter or the year. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits, and qr is the quarter of the year expressed in Roman numerals. |
| | Zw.d Format (p. 521) | Writes standard numeric data with leading 0s. |
| | ZDw.d Format (p. 522) | Writes numeric data in zoned decimal format. |
| Time | B8601TMw.d Format (p. 158) | Writes time values by using the ISO 8601 basic notation hhmmss<ffff>. |
| | B8601TXw. Format (p. 159) | Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss+\|-hhmm. |
| | B8601TZw. Format (p. 161) | Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 basic time notation hhmmss+\|–hhmm. |

| Category | Language Elements | Description |
|---|---|---|
| | E8601LZw. Format (p. 202) | 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. 204) | Writes time values by using the ISO 8601 extended notation hh:mm:ss.ffffff. |
| | E8601TXw. Format (p. 206) | Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 extended time notation hh:mm:ss+\|–hh:mm. |
| | E8601TZw.d Format (p. 208) | Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation hh:mm:ss.<fff>+\|–hh:mm. |
| | HHMMw.d Format (p. 221) | Writes time values as hours and minutes in the form hh:mm. |
| | HOURw.d Format (p. 224) | Writes time values as hours and decimal fractions of hours. |
| | MMSSw.d Format (p. 240) | Writes time values as the number of minutes and seconds since midnight. |
| | NLDATMTMw. Format (p. 293) | Writes the time portion of a SAS datetime value as a time of day that is appropriate for the current SAS locale. |
| | NLDATMTZw. Format (p. 294) | Writes the time portion of the SAS datetime value as the time of day and time zone that is appropriate for the current SAS locale. |
| | NLTIMAPw. Format (p. 416) | Writes a SAS time value as a time value with a.m. or p.m. that is appropriate for the current SAS locale. |
| | NLTIMEw. Format (p. 418) | Writes a SAS time value as a time value that is appropriate for the current SAS locale. |
| | TIMEw.d Format (p. 465) | Writes time values as hours, minutes, and seconds in the form hh:mm:ss.ss. |
| | TIMEAMPMw.d Format (p. 468) | Writes time and datetime values as hours, minutes, and seconds in the form hh:mm:ss.ss with AM or PM. |

# Dictionary

## $ASCIIw. Format

Converts native format character data to ASCII representation.

Category: Character

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

# 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

```
data one;
   input x $;
   datalines;
abc
ABC
()*
;
run;

data two;
   set one;
   put x $ascii3.;
run;
```

SAS writes the following results to the log:

```
abc
ABC
()*
```

# $BASE64Xw. Format

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

Categories: CAS
Character

Alignment: Left

## Syntax

**$BASE64X***w*.

## Syntax Description

***w***

specifies the width of the output field.

You can use the following formula to determine the width:

format-width=(variable-length+2)/3*4

When the variable-length+2 is divided by 3, the results are truncated to an integer and multiplied by 4. For example, if a variable length is 48, the width calculation is (48+2)/3*4=64.

If the format width is too small, the value is not converted. No message is written to the SAS log.

**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 email 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 /.

Here are some uses of Base 64 encoding:

- embed binary data in an XML file

- encode passwords
- encode URLs

If the encoded results contain the '=' character, it 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

```
data one;
   input x :$30.;
   datalines;
FCA01A7993BC
MyPassword
www.mydomain.com/myhiddenURL
;
run;

data two;
   set one;
   put x $base64x64.;
run;
```

SAS writes the following results to the log:

```
RkNBMDFBNzk5M0JDICAgICAgICAgICAgICAgICAg
TXlQYXNzd29yZCAgICAgICAgICAgICAgICAgICAg
d3d3Lm15ZG9tYWluLmNvbS9teWhpZGRlblVSTCAg
```

## See Also

- The LIBNAME statement option "LIBNAME Statement: XMLV2 and XML Engines" in *SAS XMLV2 and XML LIBNAME Engines: User's Guide*

**Informats:**

- "$BASE64Xw." on page 563

# $BINARYw. Format

Converts character data to binary representation.

| | |
|---|---|
| Categories: | CAS<br>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: Converting Two Characters to Binary Representation

```
data one;
   x='AB';
   put x $binary.;
run;
```

The preceding example writes this output, the binary values of A and B, to the SAS log.

```
x=0100000101000010
```

There is not a blank space between the A and B in the code example, or between their binary representations in the output.

# $CHARw. Format

Writes standard character data.

Categories: CAS
Character

Alignment: Left

# Syntax

**$CHAR***w.*

## Syntax Description

***w***

specifies the width of the output field.

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

**Range** 1–32767

# Comparisons

- The $CHAR*w*. format is not identical to the $*w*. format. The $CHAR*w*. format does not trim trailing blanks. The $*w*. format trims trailing blanks.
- 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) |

# Example:

```
data one;
   length x $5;
   input x $5. y;
   put x $char. y;

   datalines;
AB   123
;
run;
```

The preceding example specifies a length value. The example then uses the value when it writes this character data to the SAS log.

```
AB   123
```

# $CSTR Format

Looks for the last non-blank character of a character argument and passes a copy of the string with a null terminator after the last non-blank character.

Categories: CAS
Character

## Syntax

**$CSTR***w.*

### Required Argument

***w***
specifies the width of the output field.

**Default** 1

**Range** 1–32767

## Details

Use the $CSTR*w*. format to pass a character argument as a null-terminated string. $CSTR looks for the last non-blank character of a character argument and passes a copy of the string with a null terminator after the last non-blank character.

## Example

```
data _null_;
    name='XYZ';
    put name $cstr10.;
run;
```

| Statement | Value of name | Result |
|---|---|---|
| | | ----+----1 |

| Statement | Value of name | Result |
|---|---|---|
| `put x $cstr. y;` | XYZ* | `XYZ` |

**Note:** The XYZ name value represents a null character.

# $EBCDICw. Format

Converts native format character data to EBCDIC representation.

Category: Character

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Note: UTF-8 is the only supported session encoding. $EBCDIC format works only for 7-bit ASCII characters.

## 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.

On ASCII systems, the $EBCDIC*w*. format is based on the default encoding value of the LOCALE= option that is specified when SAS starts. For example, if the locale was set to en_US locale, the default encoding that is used by the $EBCDIC*w*. format is Open_ed-1047. If the locale is de_DE (German_Germany), the default encoding that is used by the $EBCDIC*w*. format is Open_ed–1141. For a list of locales and encoding values, see "Default Values for DFLANG, DATESTYLE, and PAPERSIZE System Options Based on the LOCALE= System Option" in *SAS National Language Support (NLS): Reference Guide*.

You can specify the translation table that is used to map characters between EBCDIC and ASCII by using the MAPEBCDIC2ASCII system option. For more information, see "MAPEBCDIC2ASCII=" in *SAS National Language Support (NLS): Reference Guide*.

## 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

```
data one;
   input x $;
   datalines;
ABC
{ä}
[@]
;
run;

data two;
  set one;
  y=put(x,$ebcdic3.);
  put y $HEX6.;
run;
```

***Table 2.1*** *Output Locale Values for American English, French, and German*

| Value of name | Locale Value Is en_US | Locale Value Is fr_FR | Locale Value Is de_DE |
|---|---|---|---|
| ABC | C1C2C3 | C1C2C3 | C1C2C3 |
| {ä} | C043D0 | 514354 | 43C0DC |
| [@] | AD7CBD | 9044B5 | 63B5FC |

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

# $HEXw. Format

Converts character data to hexadecimal representation.

Categories: CAS
Character

Alignment: Left

See: "$HEX Format: UNIX" in *SAS Companion for UNIX Environments*
"$HEXw.: 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 to 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

```
data one;
   x='AB';
   put x $hex4.;
run;
```

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

# $MSGCASEw. Format

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

Category: Character

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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

```
data one;
  x='sas';
```

```
  put x $msgcase.;
run;
```

| Value of name | Result |
|---|---|
| sas | SAS |

## See Also

**System Options:**

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

# $N8601Bw.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

Restrictions: This format is not supported in a DATA step that runs in CAS.
UTC time zone offset values are not supported.
Values greater than 12 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** 0–6

# 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`

**Note:** Using a month value that is greater than 12 with a datetime value causes an error. For example, the value `20181415T000000/2018-09-15T00:00:00` causes an error because the value for the month is 14.

# Examples

## Example 1: Writing a Datetime Value

This example writes a datetime value that represents June 8, 2018, 9 hours, 41 minutes, and 5.96 seconds. This newly computed datetime is a character variable, as the LENGTH statement indicates.

```
data one;
   length mynew $16;
   x='P10w';                                          /* 1 */
   y='17aug2018:9:41:05.96'dt;                        /* 2 */
   call is8601_convert('du/dt','start',x,y,mynew);
   put mynew $n8601b.;
run;
```

1 X is a duration of 10 weeks.

2 Y is a datetime value.

| Statement | Value | Result |
|---|---|---|
| put mynew $n8601b.; | 2018608094105F01 | 20180608T09410596 |

## Example 2: Writing a Duration Value

This example writes a duration value of 1 year, 2 months, 14 days, 1 hour, 34 minutes, and 32 seconds.

```
data one;
    length dur $16;
    start='01apr2017:10:41:51'dt;
    end='15jun2018:12:16:23'dt;
    call is8601_convert('dt/dt','du',start,end,dur);    /* 1 */
    put dur $n8601b.;
run;
```

1 Call IS8601_CONVERT computes the duration between datetime variables Start and End. The result is stored in the `dur` character variable.

| Statement | Value | Result |
|---|---|---|
| put dur $n8601b.; | 0001214013432FFC | P1Y2M14DT1H34M32S |

# See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# $N8601BAw.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

Restrictions: This format is not supported in a DATA step that runs in CAS.
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** 0–6

# 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`

# Examples

## Example 1: Writing a Datetime Value

This example writes a datetime value that represents June 20, 2018, 9 hours, 41 minutes, and 5.96 seconds. This newly computed datetime is a character variable, as the LENGTH statement indicates.

```
data one;
```

```
   length mynew $16;
   x='P10w';                                          /* 1 */
   y='17aug2018:9:41:05.96'dt;                        /* 2 */
   call is8601_convert('du/dt','start',x,y,mynew);
   put mynew $n8601ba.;
run;
```

1 X is a duration of 10 weeks.

2 Y is a datetime value.

| Statement | Value | Result |
|---|---|---|
| `put mynew $n8601ba.;` | 2018608094105F01 | `20180608T09410596` |

## Example 2: Writing a Duration Value

This example writes a duration value of 1 year, 2 months, 14 days, 1 hour, 34 minutes, and 32 seconds.

```
data one;
    length dur $16;
    start='01apr2017:10:41:51'dt;
    end='15jun2018:12:16:23'dt;
    call is8601_convert('dt/dt','du',start,end,dur);   /* 1 */
    put dur $n8601ba.;
run;
```

1 Call IS8601_CONVERT computes the duration between datetime variables Start and End. The result is stored in the `dur` character variable.

| Statement | Value | Result |
|---|---|---|
| `put dur $n8601ba.;` | 0001214013432FFC | `P00010214T013432` |

# See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# $N8601Ew.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

Restrictions: This format is not supported in a DATA step that runs in CAS.
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** 0–6

# 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`

# Examples

## Example 1: Writing a Datetime Value

This example writes a datetime value that represents June 20, 2018, 9 hours, 41 minutes, and 5.96 seconds. This newly computed datetime is a character variable, as the LENGTH statement indicates.

```
data one;
   length mynew $16;
   x='P10w';                                          /* 1 */
   y='17aug2018:9:41:05.96'dt;                        /* 2 */
   call is8601_convert('du/dt','start',x,y,mynew);
   put mynew $n8601e.;
run;
```

1 X is a duration of 10 weeks.

2 Y is a datetime value.

| Statement | Value | Result |
|---|---|---|
| `put mynew $n8601e.;` | 2018608094105F01 | `2018-06-08T09:41:05.96` |

## Example 2: Writing a Duration Value

This example writes a duration value of 1 year, 2 months, 14 days, 1 hour, 34 minutes, and 32 seconds.

```
data one;
    length dur $16;
    start='01apr2017:10:41:51'dt;
    end='15jun2018:12:16:23'dt;
    call is8601_convert('dt/dt','du',start,end,dur);    /* 1 */
    put dur $n8601e.;
run;
```

1 Call IS8601_CONVERT computes the duration between datetime variables Start and End. The result is stored in the `dur` character variable.

| Statement | Value | Result |
|---|---|---|
| `put dur $n8601e.;` | 0001214013432FFC | `P1Y2M14DT1H34M32S` |

# See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# $N8601EAw.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<br>ISO 8601 |
| Alignment: | Left |
| Restrictions: | This format is not supported in a DATA step that runs in CAS.<br>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** | 0–6 |

## 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`

# Examples

## Example 1: Writing a Datetime Value

This example writes a datetime value that represents June 20, 2018, 9 hours, 41 minutes, and 5.96 seconds. This newly computed datetime is a character variable, as the LENGTH statement indicates.

```
data one;
   length mynew $16;
   x='P10w';                                          /* 1 */
   y='17aug2018:9:41:05.96'dt;                        /* 2 */
   call is8601_convert('du/dt','start',x,y,mynew);
   put mynew $n8601ea.;
run;
```

1 X is a duration of 10 weeks.

2 Y is a datetime value.

| Statement | Value | Result |
|---|---|---|
| `put mynew $n8601ea.;` | 2018608094105F01 | `2018-06-08T09:41:05.96` |

## Example 2: Writing a Duration Value

This example writes a duration value of 1 year, 2 months, 14 days, 1 hour, 34 minutes, and 32 seconds.

```
data one;
    length dur $16;
    start='01apr2017:10:41:51'dt;
    end='15jun2018:12:16:23'dt;
    call is8601_convert('dt/dt','du',start,end,dur);  /* 1 */
    put dur $n8601ea.;
run;
```

1 Call IS8601_CONVERT computes the duration between datetime variables Start and End. The result is stored in the `dur` character variable.

| Statement | Value | Result |
|---|---|---|
| `put dur $n8601ea.;` | P0001-02-14T01:34:32 | `P1Y2M14DT1H34M32S` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# $N8601EHw.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

Restrictions: This format is not supported in a DATA step that runs in CAS.
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** 0–6

## 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.

# Examples

## Example 1: Writing a Datetime Value

This example writes a datetime value that represents June 20, 2018, 9 hours, 41 minutes, and 5.96 seconds.

```
data one;
   length mynew $16;
   x='P10w';                                        /* 1 */
   y='17aug2018:9:41:05.96'dt;                      /* 2 */
   call is8601_convert('du/dt','start',x,y,mynew);
   put mynew $n8601eh.;
run;
```

1 X is a duration of 10 weeks.

2 Y is a datetime value.

| Statement | Value | Result |
|---|---|---|
| `put mynew $n8601eh.;` | 2018608094105F01 | `2018-06-08T09:41:05.96` |

## Example 2: Writing a Duration Value

This example writes a duration value of 1 year, 2 months, 14 days, 1 hour, 34 minutes, and 32 seconds.

```
data one;
   length dur $16;
   start='01jun2017:10:41:51'dt;
   end='15jun2018:12:16:23'dt;
   call is8601_convert('dt/dt','du',start,end,dur);  /* 1 */
   put dur $n8601eh.;
run;
```

1 Call IS8601_CONVERT computes the duration between datetime variables Start and End. The result is stored in the `dur` character variable.

In the table, omitted components of datetimes are represented by a hyphen (-).

| Statement | Value | Result |
|---|---|---|
| `put dur $n8601eh.;` | 0001214013432FFC | `P0001---14T01:34:32` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# $N8601EXw.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

Restrictions: This format is not supported in a DATA step that runs in CAS.
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** 0–6

# 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.

# Examples

## Example 1: Writing a Datetime Value

This example writes a datetime value that represents June 20, 2018, 9 hours, 41 minutes, and 5.96 seconds. This newly computed datetime is a character variable, as the LENGTH statement indicates.

```
data one;
   length mynew $16;
   x='P10w';                                        /* 1 */
   y='17aug2018:9:41:05.96                          /* 2 */

   put mynew $n8601ex.;
run;
```

1 X is a duration of 10 weeks.

2 Y is a datetime value.

| Statement | Value | Result |
|---|---|---|
| `put mynew $n8601ex.;` | 2018608094105F01 | `2018-06-08T09:41:05.9` |

## Example 2: Writing a Duration Value

This example writes a duration value of 1 year, 2 months, 14 days, 1 hour, 34 minutes, and 32 seconds.

```
data one;
   length dur $16;
   start='01jun2017:10:41:51'dt;
   end='15jun2017:12:16:23'dt;
   call is8601_convert('dt/dt','du',start,end,dur);  /* 1 */
   put dur $n8601ex.;
run;
```

1 Call IS8601_CONVERT computes the duration between datetime variables Start and End. The result is stored in the `dur` character variable.

In the table, omitted components of datetimes are represented by **x**.

| Statement | Value | Result |
|---|---|---|
| `put dur $n8601ex.;` | 0001214013432FFC | `P0001-02-14T01:34:3`<br>2 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# $N8601Hw.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

Restrictions: This format is not supported in a DATA step that runs in CAS.
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** 0–6

# 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.

# Examples

## Example 1: Writing a Datetime Value

This example writes a datetime value that represents June 20, 2018, 9 hours, 41 minutes, and 5.96 seconds. This newly computed datetime is a character variable, as the LENGTH statement indicates.

```
data one;
   length mynew $16;
   x='P10w';                                        /* 1 */
   y='17aug2018:9:41:05.96'dt;                      /* 2 */
   call is8601_convert('du/dt','start',x,y,mynew);
   put mynew $n8601h.;
run;
```

1 X is a duration of 10 weeks.

2 Y is a datetime value.

| Statement | Value | Result |
|---|---|---|
| `put mynew $n8601e.;` | 2018608094105F01 | `2018-06-08T09:41:05.96` |

## Example 2: Writing a Duration Value

This example writes a duration value of 1 year, 2 months, 14 days, 1 hour, 34 minutes, and 32 seconds.

```
data one;
   length dur $16;
   start='01jun2017:10:41:51'dt;
   end='15jun2017:12:16:23'dt;
   call is8601_convert('dt/dt','du',start,end,dur);  /* 1 */
   put dur $n8601h.;
run;
```

1 Call IS8601_CONVERT computes the duration between datetime variables Start and End. The result is stored in the `dur` character variable.

| Statement | Value | Result |
|---|---|---|
| `put dur $n8601h.;` | 0001214013432FFC | `P1Y2M14DT1H34M32S` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# $N8601Xw.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

Restrictions: This format is not supported in a DATA step that runs in CAS.
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** 0–6

# 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

This example writes a datetime value that represents June 5, 2018, 10 hours, 15 minutes, and 20 seconds. This newly computed datetime is a character variable, as the LENGTH statement indicates.

```
data one;
   input y mo d h min s;
   length dt $16;
   call is8601_convert('dt6','start',y,mo,d,h,min,s,dt);
   put dt $n8601x.;
   datalines;
2018 6 . 10 15 20
2018 . 5 10 15 20
;
run;
```

In the table, omitted components of datetimes are represented by **x**.

| Statement | Value | Result |
|---|---|---|
| `put dt $n8601x.;` | 20186FF101520001 | `2018-06-xxT10:15:20` |
| `put dt $n8601x.;` | 2018F05101520001 | `2018-xx-05T10:15:20` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# $OCTALw. Format

Converts character data to octal representation.

Categories: CAS
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
```

# $QUOTEw. Format

Writes data values that are enclosed in double quotation marks.

Categories: CAS
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 plus 2

**Range** 1–32767

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

# Details

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

- 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 behaves in one of these ways:
  - encloses the data value in double quotation marks
  - leaves the single quotation mark
- If your data value begins and ends with single quotation marks, and the value contains double quotation marks, SAS behaves in one of these ways:
  - encloses the data value in double quotation marks
  - duplicates the double quotation marks that are found in the data value
  - leaves 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 behaves in one of these ways:
  - encloses the value in double quotation marks
  - leaves 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 behaves in one of these ways:
  - encloses the value in double quotation marks
  - duplicates the double quotation marks that are found in the data value
  - leaves 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 as fits in the field.

  The result contains leading and trailing quotation marks, and the value is truncated to ensure enough room in the field for the quotation marks. For example, if the specified value is ABCDE and you specify $QUOTE5., the DE in the value is truncated and the result is `"ABC"`.

  **Note:** An embedded quotation mark must be enclosed within additional quotation marks. If you specify a value of A"B, then $QUOTE5. truncates the B in the value and writes `"A"""`.

## Example

```
data test;
   input original $19.;
   format original $quote20.;
   datalines;
SAS
SAS's
'ad"verb"'
'ad''verb'
'"ad"'"verb"'
deoxyribonucleotide
;
run;
proc print noobs; run;
```

| Value of original | Result |
|---|---|
| | `----+----1----+----2` |
| SAS | `"SAS"` |
| SAS's | `"SAS's"` |
| 'ad"verb"' | `"'ad""verb""'"` |
| 'ad"verb' | `"'ad''verb'"` |
| '"ad"""verb"' | `----+----1----+----2`<br>`"'""ad""'""verb""'"` |
| deoxyribonucleotide | `"deoxyribonucleotid"` [1] |

1 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.

# $REVERJw. Format

Writes character data in reverse order and preserves blanks.

Categories: CAS
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

```
data _null_;
   x='ABCD   ';
   put x $reverj7.;
   y='   ABCD';
   put y $reverj7.;
run;
```

| Statement | Name[1] | Result |
|---|---|---|
| | | `----+----1` |
| put x $reverj7.; | `ABCD###` | `DCBA` |
| put y $reverj7.; | `###ABCD` | `DCBA` |

1 The character # represents a blank space.

# $REVERSw. Format

Writes character data in reverse order and left-aligns

Categories: CAS
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

```
data _null_;
   x='ABCD   ';
   put x $revers7.;
   y='   ABCD';
   put y $revers7.;
run;
```

| Statement | Name[1] | Result |
|---|---|---|
| | | `----+----1` |
| put x $revers7.; | `ABCD###` | `DCBA` |
| put y $revers7.; | `###ABCD` | `DCBA` |

1 The character # represents a blank space.

# $UPCASEw. Format

Converts character data to uppercase.

Categories: CAS
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

```
data one;
   name='coxe-ryan';
   put name $upcase9.;
run;
```

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

# $UUIDw. Format

Converts character data to the Universally Unique Identifier (UUID) format.

Categories: CAS
Character

Alignment: Left

## Syntax

**$UUID***w*.

### Arguments

***w***

specifies the width of the output field.

**Range** 1–200

## Details

The value to be formatted must be a 16-byte binary value. The $UUID format displays the binary data with hexadecimal characters and hyphens as delimiters. If you specify the value in a format other than hexadecimal, the returned UUID is the hexadecimal representation of the original characters.

**Note:** In this example, the binary value is set via a hexadecimal literal.

## Example

```
data _null_;
   length x $32 y $36;
   x='1548611fb8cb6a47a721a6fd284e74f2'x;
   y=put(x, $uuid36.);
   put y;
run;
```

| Value of x | Result |
|---|---|
| 1548611fb8cb6a47a721a6fd284e74f2 | `1548611f-b8cb-6a47-a721-a6fd284e74f2` |

# $VARYINGw. Format

Writes character data of varying length.

Valid in: DATA step

Category: Character

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

# 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

```
data one;
   infile datalines truncover;
   input @1 name & $12.  @14 varlen;
```

```
   datalines;
New York     8
Toronto      7
Buenos Aires 12
Tokyo        5
;
run;

proc print;
run;

data _null_;
   set one;
   put name $varying12. varlen;
run;
```

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 [1] | Result |
|---|---|
| | `----+----1----+----2----+` |
| **New York 8** | `New York` |
| **Toronto 7** | `Toronto` |
| **Buenos Aires 12** | `Buenos Aires` |
| **Tokyo 5** | `Tokyo` |

1 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.

| Value of varlen[1] | Result |
|---|---|
| | `----+----1----+----2----+` |

| Value of varlen[1] | Result |
|---|---|
| New York | `New York` |
| Toronto | `Toronto` |
| Buenos Aires | `Buenos Aires` |
| Tokyo | `Tokyo` |

1 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** 8, if the length of the variable is undefined. Otherwise, the default is the length of the variable.

**Range** 1–32767

## Comparisons

The $*w*. format is not identical to the $CHAR*w*. format. The $*w*. format trims trailing blanks. The $CHAR*w*. format does not trim trailing blanks.

The $*w*. format and the $CHAR*w*. format 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

```
data one;
   input name $char5.;
   datalines;
 Cary
Tokyo
 ;
 run;
 data two;
    set one;
    put @name $5.;
    put name $ 10-15;
 run;
```

| Value of name[1] | Result |
|---|---|
| | `----+----1----+----2` |
| #Cary | `Cary` |
| Tokyo | `Tokyo` |

1 The character # represents a blank space.

# BESTw. Format

SAS chooses the best notation.

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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, use a field width of at least 7 to avoid excessive rounding. If you print numbers between 0 and –.01 exclusively, 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 attempts to write numbers that balance the conflicting requirements of readability, precision, and brevity. Here are several rules:

- Values are written with the maximum readable precision, as determined by the width. The maximum precision for floating-point numbers might be limited to 14 or 15 digits, 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.
- Extreme values and values with leading or trailing 0s might be written in scientific notation to fit into the specified width, to increase the precision, or to simplify the magnitude of the number. Extremely small values might be written as 0 if the width is too small for scientific notation.
- Trailing 0s are not written.
- If a value cannot be represented in either decimal or scientific notation in the width that is specified, the output field is filled with asterisks.
- The behavior of the BEST*w*. format is affected by the setting of the DECIMALCONV option. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*. These rules are generally applicable regardless of the option setting.

# 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 are printed 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

```
data one;
   x=1257000;
   put x best6.;
   put x best3.;
run;
```

The following statements produce these results:

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `x=1257000;`<br>`put x best6.;` | `1.26E6` |
| `x=1257000;`<br>`put x best3.;` | `1E6` |

## See Also

**Formats:**

- “BESTDw.p” on page 143

# BESTDw.p Format

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

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV=” in *SAS System Options: Reference*.

## 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** *p* 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 are printed 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

```
data one;
   input x;
```

```
   datalines;
12345
123.45
1.2345
.12345
;
run;

data _null_;
   set one;
   put x bestd9.;
run;
```

| Variable | Result |
|---|---|
| | ----+----1----+ |
| 12345 | 12345 |
| 123.45 | 123.4500 |
| 1.2345 | 1.2345 |
| .12345 | 0.12345 |

## See Also

**Formats:**

- “BESTw.” on page 141
- “Dw.p” on page 166

# BINARYw. Format

Converts numeric values to binary representation.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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

```
data one;
   input x;
   datalines;
123.45
123
-123
;
run;

data two;
   set one;
   put x binary8.;
run;
```

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

# B8601DAw. Format

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

Categories: Date

ISO 8601

Alignment: Left

Restrictions: This format is not supported in a DATA step that runs in CAS.
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

```
data one;
   x='15may2018'd;
   put x b8601da.;
run;
```

| Value of x | Result |
|---|---|
| 21319 | 20180515 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# B8601DNw. Format

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

| | |
|---|---|
| Categories: | Date<br>Date and Time<br>ISO 8601 |
| Alignment: | Left |
| Restrictions: | This format is not supported in a DATA step that runs in CAS.<br>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

```
data one;
   x='15may2018:14:52:22'dt;
   put x b8601dn.;
run;
```

| Value of x | Result |
|---|---|
| 21319 | 20180515 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# B8601DTw.d Format

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

Categories: Date and Time
ISO 8601

Alignment: Left

Restrictions: This format is not supported in a DATA step that runs in CAS.
UTC time zone offset values are not supported.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

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

```
data one;
   x='15may2018:14:52:22'dt;
   put x b8601dt.;
run;
```

| Value of x | Result |
|---|---|
| ---- +----1 | |
| 1842015142 | `20180515T145222` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# B8601DXw. Format

Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone basic notation *yyyymmdd*T*hhmmss +hhmm*.

| | |
|---|---|
| Categories: | Date and Time<br>ISO 8601 |
| Alignment: | Left |
| Restriction: | This format is not supported in a DATA step that runs in CAS. |
| Supports: | ISO 8601 Element 5.4.1, complete representation |

## Syntax

**B8601DX***w*.

### Syntax Description

***w***
specifies the width of the output field.

**Default** 26

**Range** 20–35

## Details

UTC values specify a date and a time that are based on the zero meridian in Greenwich, England. Using this format, SAS converts a datetime value to the UTC value and determines the user local date and time by using the value of the TIMEZONE= system option. If the TIMEZONE= option is not set, then the user local date and time are based on the local date and time. The B8601DX format writes SAS datetime values by using the following ISO 8601 basic datetime notation:

- *yyyymmdd*T*hhmmss+hhmm*

*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.

+|–*hhmm*
is an hour and minute signed offset from zero meridian time. 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.

# Example

The first example uses the local time to determine the time and the time zone offset. The second example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

| Statement | Result |
|---|---|
| `data _null_ ;`<br>`t='01Feb2013T12:34:56'dt ;`<br>`put t b8601dx.;`<br>`run;` | `20130201T073456-0500` |
| `options timezone='America/Adak';`<br>`data _null_ ;`<br>`t='01Feb2013T12:34:56'dt ;`<br>`put t b8601dx.;`<br>`run;` | `20130201T023456-1000` |

# See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# B8601DZw. 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<br>ISO 8601 |
| Alignment: | Left |
| Restriction: | This format is not supported in a DATA step that runs in CAS. |
| 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 these 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. Then, the time zone offset is processed as the time or datetime for the zero meridian (Greenwich, England). The B8601DZ format always writes the datetime value by using the zero meridian offset value of +0000. To write a datetime that uses a time zone offset other than +0000, see "B8601LZw." on page 156.

**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

```
data _null_;
   offset_added=input('20180601T123456+0500',b8601dz.);
   nooffset=input('20180601T123456Z',b8601dz.);
   put offset_added b8601dz.;
   put nooffset b8601dz.;
run;
```

```
20180601T073456+0000
20180601T123456+0000
```

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# B8601LXw. Format

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

Categories: Date and Time

ISO 8601

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

Supports: ISO 8601 Elements 5.3.3 and 5.3.4.2

# Syntax

**B8601LX***w*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 26

**Range** 20–35

# Details

The B8601LX format writes datetime values without making any adjustments, and appends the UTC time zone offset for the local SAS session by using the ISO 8601 basic datetime notation:

- *yyyymmdd*T*hhmmss*+|–*hhmm*

*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.

+|–*hhmm*
is an hour and minute signed offset from zero meridian time. 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.

## Example

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

```
blx='01Feb2013T12:34:56'dt;
put blx b8601lx.;
```

| Value of blx | Result |
|---|---|
| 1675341296 | 20130201T123456-0500 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# B8601LZw. 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

Restriction: This format is not supported in a DATA step that runs in CAS.

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

```
data one;
   x='15:32't;
   put x b8601lz.;
run;
```

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

| Value of x | Result |
|---|---|
| 21319 | 153200-0400 |

# See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 45

# B8601TMw.d Format

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

Categories: Time
ISO 8601

Alignment: Left

Restrictions: This format is not supported in a DATA step that runs in CAS.
UTC time zone offset values are not supported.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

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

```
data one;
   x='15:32't;
   put x b8601tm.;
run;
```

| Value of x | Result |
|---|---|
| 55920 | `153200` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# B8601TXw. Format

Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 basic time notation *hhmmss*+|-*hhmm*.

| | |
|---|---|
| Categories: | Time<br>ISO 8601 |
| Alignment: | Right |
| Restriction: | This format is not supported in a DATA step that runs in CAS. |
| Supports: | ISO 8601 Elements 5.3.3 and 5.3.4 |

## Syntax

**B8601TX***w*.

### Syntax Description

***w***
specifies the width of the output field.

**Default** 14

**Range** 9–20

# Details

UTC values specify a time based on the zero meridian in Greenwich, England. Using this format, SAS converts a time value to the UTC value and determines the user local time by using the TIMEZONE= system option. If the TIMEZONE= option is not set, then the user local time is based on the local time. The B8601TX format writes SAS datetime values by using the following ISO 8601 basic time 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. 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 240000. If the B8601TX 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

The first example uses the local time to determine the time and the time zone offset. The second example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

| Statement | Result |
|---|---|
| `data _null_;`<br>`t='12:34:56't;`<br>`put t b8601tx.;`<br>`run;` | `073456-0500` |

| Statement | Result |
| --- | --- |
| `options timezone='America/Adak';`<br>`data _null_ ;`<br>`t='12:34:56't;`<br>`put t b8601tx.;`<br>`run;` | `023456-1000` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# B8601TZw. 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: Time
ISO 8601

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

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 these 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

```
data one;
```

```
   x='15:32't;
   put x b8601tz.;
run;
```

| Value for x | Result |
|---|---|
| 55920 | 153200+0000 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# COMMAw.d Format

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

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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

```
data one;
   input x;
   datalines;
23451.23
123451.234
;
run;

data two;
   set one;
   put x comma10.2;
run;
```

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

## See Also

**Formats:**


- "COMMAXw.d" on page 165
- "DOLLARw.d" on page 180


# COMMAXw.d Format

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

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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

```
data one;
   input x;
   datalines;
23451.23
123451.234
;
run;

data two;
   set one;
   put x commax10.2;
run;
```

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

# Dw.p Format

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

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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** 0–16

**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

```
data one;
   input x;
   datalines;
12345
1234.5
123.45
12.345
1.2345
.12345
;
run;

data two;
   set one;
   put x d10.4;
run;
```

| 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” on page 143

# DATEw. Format

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

Categories: CAS
Date

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 four-digit year without a separator between the day, month, and year. Use a width of 11 to print a four-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*.

*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

```
data one;
/* create SAS date with date literal */
   x='15mar2018'd;
```

```
      put x date5.;
      put x date6.;
      put x date7.;
      put x date8.;
      put x date9.;
      put x date11.;
   run;
```

The example table uses the input value of 21258, which is the SAS date value that corresponds to March 15, 2018.

| SAS Statement | Result |
|---|---|
| | ----+----1----+ |
| put x date5.; | 15MAR |
| put x date6.; | 15MAR |
| put x date7.; | 15MAR18 |
| put x date8.; | 15MAR18 |
| put x date9.; | 15MAR2018 |
| put x date11.; | 15-MAR-2018 |

## See Also

**Functions:**

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

**Formats:**

- "NLDATEw."

**Informats:**

- "DATEw." on page 617

# DATEAMPMw.d Format

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

Categories: CAS
Date and Time

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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** *d* 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*.

*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

```
data one;
   x='14mar2018:-1:34:27'dt;
   put x dateampm.;
   put x dateampm7.;
   put x dateampm10.;
   put x dateampm13.;
   put x dateampm22.2;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+----2----+` |
| `put x dateampm.;` | `13MAR18:10:25:33 PM` |
| `put x dateampm7.;` | `13MAR18` |
| `put x dateampm10.;` | `13MAR18:22` |
| `put x dateampm13.;` | `13MAR18:10 PM` |
| `put x dateampm22.2;` | `13MAR18:10:25:33.00 PM` |

## See Also

**Formats:**

- “DATETIMEw.d” on page 172

# DATETIMEw.d Format

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

Categories: CAS
Date and Time

Alignment: Right

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

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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** *d* 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

```
data one;
   x='15mar2018:-1:34:27'dt;
   put x datetime.;
   put x datetime7.;
   put x datetime12.;
   put x datetime18.;
   put x datetime18.1;
   put x datetime19.;
   put x datetime20.1;
   put x datetime21.2;
run;
```

| SAS Statement | Result |
|---|---|
| | ----+----1----+----2----+ |
| put x datetime.; | 14MAR18:22:25:33 |
| put x datetime7.; | 14MAR18 |
| put x datetime12.; | 14MAR18:22 |
| put x datetime18.; | 14MAR18:22:25:33 |
| put x datetime18.1; | 14MAR18:22:25:33.0 |
| put x datetime19.; | 14MAR2018:22:25:33 |
| put x datetime20.1; | 14MAR2018:22:25:33.0 |
| put x datetime21.2; | 14MAR2018:22:25:33.00 |

# See Also

**Formats:**

- “DATEw.” on page 169
- “TIMEw.d” on page 465

**Functions:**

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

**Informats:**

- “DATEw.” on page 617
- “DATETIMEw.” on page 618

■ “TIMEw.” on page 702

# DAYw. Format

Writes date values as the day of the month.

Categories: CAS
Date

Alignment: Right

## Syntax

**DAY***w*.

### Syntax Description

***w***
specifies the width of the output field.

**Default** 2

**Range** 2–32

## Example

```
data one;
   x='15jun2018'd;
   put x day2.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+` |
| `put x day2.;` | `15` |

# DDMMYYw. 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 two or four digits.

Categories: CAS
Date

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 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

```
data one;
   x='15may2018'd;
   put x ddmmyy5.;
   put x ddmmyy6.;
   put x ddmmyy7.;
   put x ddmmyy8.;
   put x ddmmyy10.;
run;
```

| Value of x | Result |
|---|---|
| | `----+----1` |
| put x ddmmyy5.; | `15/05` |
| put x ddmmyy6.; | `150518` |
| put x ddmmyy7.; | ` 150518` |
| put x ddmmyy8.; | `15/05/18` |
| put x ddmmyy10.; | `15/05/2018` |

## See Also

**Formats:**

- “DATEw.” on page 169
- “DDMMYYxw.” on page 177
- “MMDDYYw.” on page 235
- “YYMMDDw.” on page 498

**Functions:**

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

**Informats:**

- “DATEw.” on page 617
- “DDMMYYw.” on page 621
- “MMDDYYw.” on page 649
- “YYMMDDw.” on page 720

# DDMMYYxw. 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. The special character can be a blank character, colon (:), hyphen (-),no separator, period (.), or slash (/). The year can be either two or four digits.

Categories: CAS
Date

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 values are valid 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 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 from 2 to 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

```
data one;
   x='15jun2018'd;
   put x ddmmyyc5.;
   put x ddmmyyd8.;
   put x ddmmyyp10.;
   put x ddmmyyn8.;
run;
```

| SAS Statement | Result |
|---|---|
| | ----+----1----+ |
| put x ddmmyyc5.; | 15:06 |
| put x ddmmyyd8.; | 15-06-18 |
| put x ddmmyyp10.; | 15.06.2018 |
| put x ddmmyyn8.; | 15062018 |

# See Also

**Formats:**


- "DATEw." on page 169
- "DDMMYYw." on page 175
- "MMDDYYxw." on page 237
- "YYMMDDxw." on page 501


**Functions:**

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

**Informats:**


- "DDMMYYw." on page 621

# DOLLARw.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.

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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 information, see "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

```
data one;
   netpay=1254.71;
   put netpay dollar10.2;
run;
```

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

## See Also

**Formats:**

- "COMMAw.d" on page 163
- "DOLLARXw.d" on page 181

# DOLLARXw.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.

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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** *d* 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. For more information, see "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 COMMAX*w.d* format does not write a leading dollar sign.

# Example

```
data one;
```

```
   netpay=1254.71;
   put netpay dollarx10.2;
run;
```

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

## See Also

**Formats:**

- “COMMAXw.d” on page 165
- “DOLLARw.d” on page 180

# DOWNAMEw. Format

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

Categories: CAS
Date

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. format prints the first two letters of the day name.

## Example

```
data one;
   mydate='28MAR2018'd;
   put mydate downname.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydate downname.;` | `Wednesday` |

## See Also

**Formats:**

- "WEEKDAYw." on page 482

# DTDATEw. Format

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

Categories: Date and Time
Date

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 four-digit year.

# Details

The DTDATE*w*. format writes SAS date values in the form *ddmmmyy* or *ddmmmyyyy*:

*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.

# 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

```
data one;
   mydt='28mar2018 13:45:14'dt;
   put mydt dtdate.;
   put mydt dtdate9.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydt dtdate.;` | `28MAR18` |
| `put mydt dtdate9.;` | `28MAR2018` |

## See Also

**Formats:**

- "DATEw." on page 169

# DTMONYYw. Format

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

Categories: CAS
Date
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*:

*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

```
data one;
   mydt='28mar2018 13:45:14'dt;
   put mydt dtmonyy.;
   put mydt dtmonyy5.;
   put mydt dtmonyy6.;
   put mydt dtmonyy7.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydt dtmonyy.;` | `MAR18` |
| `put mydt dtmonyy5.;` | `MAR18` |
| `put mydt dtmonyy6.;` | ` MAR18` |
| `put mydt dtmonyy7.;` | `MAR2018` |

## See Also

**Formats:**

- “DATETIMEw.d” on page 172
- “MONYYw.” on page 247

# DTWKDATXw. 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*).

Categories: CAS
Date
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*.

*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

```
data one;
   mydt='28mar2018 13:45:14'dt;
   put mydt dtwkdatx.;
   put mydt dtwkdatx3.;
   put mydt dtwkdatx8.;
   put mydt dtwkdatx25.;
```

```
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+----2----+` |
| `put mydt dtwkdatx.;` | `Wednesday, 28 March 2018` |
| `put mydt dtwkdatx3.;` | `Wed` |
| `put mydt dtwkdatx8.;` | `Wed` |
| `put date dtwkdatx25.;` | `Wednesday, 28 Mar 2018` |

## See Also

**Formats:**

- "DATETIMEw.d" on page 172
- "WEEKDATXw." on page 480

# DTYEARw. Format

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

Categories: CAS
Date
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

```
data one;
   mydt='28mar2018 13:45:14'dt;
   put mydt dtyear.;
   put mydt dtyear2.;
   put mydt dtyear3.;
   put mydt dtyear4.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydt dtyear.;` | `2018` |
| `put mydt dtyear2.;` | `18` |
| `put mydt dtyear3.;` | ` 18` |
| `put mydt year4.;` | `2018` |

## See Also

**Formats:**

- “DATETIMEw.d” on page 172
- “YEARw.” on page 495

# DTYYQCw. Format

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

Categories: CAS
Date
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

```
data one;
   mydt='28mar2018 13:45:14'dt;
   put mydt dtyyqc.;
   put mydt dtyyqc4.;
   put mydt dtyyqc5.;
   put mydt dtyyqc6.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydt dtyyqc.;` | `18:1` |
| `put mydt dtyyqc4.;` | `18:1` |
| `put mydt dtyyqc5.;` | `18:1` |
| `put mydt dtyyqc6.;` | `2018:1` |

# See Also

**Formats:**

■ “DATETIMEw.d” on page 172

# Ew. Format

Writes numeric values in scientific notation.

Categories: CAS
Numeric

Alignment: Right

## 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

```
data one;
   input value;
   datalines;
1257
-1257
;
run;

data two;
   set one;
   put value e10.;
run;
```

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

# E8601DAw. Format

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

Categories: CAS
Date
ISO 8601

Alignment: Left

Alias: IS8601DA*w*.

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

```
data one;
   mydt='15sep2018'd;
   put mydt e8601da.;
run;
```

| Value for mydt | Result |
|---|---|
| 21442 | 2018-09-15 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601DNw. Format

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

Categories: CAS
Date
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 output field.

**Default** 10

**Requirement** The width of the output 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

```
data one;
   mydt='15sep2018 14:52:22'dt;
   put mydt e8601dn.;
run;
```

| Value for mydt | Result |
|---|---|
| 1852642342 | `2018-09-15` |

# See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601DTw.d Format

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

Categories: CAS
Date and Time

ISO 8601

Alignment: Left

Alias: IS8601DT*w.d*

Restriction: UTC time zone offset values are not supported.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

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** 16–26

***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.

**Note:** If you specify a width of 16, SAS assumes that the value for seconds is 0 and omits them from the output.

## Example

```
data one;
   mydt='15sep2018 14:52:22'dt;
   put mydt e8601dt25.3;
run;
```

| Value of mydt | Result |
|---|---|
| 1852642342 | 2018-09-15T14:52:22.000 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601DXw. Format

Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, writes the local date and time by using the ISO 8601 datetime and time zone extended notation *yyyy-mm-dd*T*hh:mm:ss+hh:mm*.

Categories: Date and Time
ISO 8601

Alignment: Left

Supports: ISO 8601 Element 5.4.1, complete representation

## Syntax

**E8601DX***w*.

## Syntax Description

***w***
specifies the width of the output field.

**Default** 26

**Range** 20–35

# Details

UTC values specify a date and time that are based on the zero meridian in Greenwich, England. Using this format, SAS converts a datetime value to the UTC value and determines the user local date and time by using the value of the TIMEZONE= system option. If the TIMEZONE= option is not set, the user local date and time are based on the local date and time. The E8601DX format writes SAS datetime values by using this ISO 8601 basic datetime notation:

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

*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.

+|–*hh:mm*
is an hour and minute signed offset from zero meridian time. The offset must be +|–*hh:mm* (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, +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.

# Example

The first example uses the local time to determine the time. The second example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

| Statement | Result |
| --- | --- |
| `data _null_ ;`<br>`t='15Sep2018T12:34:56'dt ;`<br>`put t e8601dx.;`<br>`run;` | `2018-09-15T08:34:56-04:00` |
| `options timezone='America/Adak';`<br>`data _null_ ;`<br>`t='15Sep2018T12:34:56'dt ;`<br>`put t e8601dx.;`<br>`run;` | `2018-09-15T03:34:56-09:00` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601DZw. Format

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

Categories: CAS
Date and Time
ISO 8601

Alignment: Left

Alias: IS8601DZ*w*.

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 these 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." on page 202.

**Restriction:** The shorter form +00 is not supported.

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

# Example

```
data one;
   mydt='15sep2018 14:52:22'dt;
   put mydt e8601dz.;
run;
```

| Value of mydt | Result |
| --- | --- |
| 1852642342 | `2018-09-15T14:52:22+00:00` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601LXw. Format

Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation *yyyy-mm-dd*T*hh:mm:ss*+|–*hh:mm*.

Categories: Date and Time
ISO 8601

Alignment: Right

Supports: ISO 8601 Elements 5.3.3 and 5.3.4.2

## Syntax

**E8601LX***w*

### Syntax Description

***w***
specifies the width of the output field.

**Default** 26

**Range** 20–35

## Details

The E8601LX format writes datetime values without making any adjustments, and appends the UTC time zone offset for the local SAS session by using this ISO 8601 basic datetime notation:

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

*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.

+|–*hh:mm*
is an hour and minute signed offset from zero meridian time. The offset must be +|–*hh:mm* (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, +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.

## Example

This example writes the time for the Eastern time zone:

```
data one;
   mydt='15sep2018 T12:34:56'dt;
   put mydt e8601lx.
run;
```

| Value of mydt | Result |
|---|---|
| 1852634096 | `2018-09-15T12:34:56-05:00` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601LZw. 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: CAS

Time
ISO 8601

Alignment: Left

Alias: IS8601LZ*w*.

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 these 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. 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.

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 Time in the U.S., and the time value is 45824, SAS would write 12:43:44–04:00 or 16:43:44Z. 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 example writes the time for the Eastern Time zone:

```
data one;
   mytime='14:52:26't;
   put mytime e8601lz.;
run;
```

| Value of mytime | Result |
|---|---|
| 53546 | `14:52:26-05:00` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601TMw.d Format

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

Categories: CAS

Time

ISO 8601

Alignment: Left

Alias: IS8601TM*w.d*

Restriction: UTC time zone offset values are not supported.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

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

```
data one;
   mytime='14:52:26't;
   put mytime e8601tm.;
run;
```

| Value of mytime | Result |
|---|---|
| 53546 | 14:52:26 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601TXw. Format

Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, writes the local time by using the ISO 8601 extended time notation *hh:mm:ss*+|–*hh:mm*.

Categories: Time
ISO 8601

Alignment: Right

Supports: ISO 8601 Elements 5.3.3 and 5.3.4

## Syntax

**E8601TX***w*.

## Syntax Description

***w***
specifies the width of the output field.

**Default** 14

**Range** 9–20

# Details

UTC values specify a time based on the zero meridian in Greenwich, England. Using this format, SAS converts a time value to the UTC value and determines the user local time by using the value of the TIMEZONE= system option. If the TIMEZONE= option is not set, the user local time is based on the local time. The E8601TX format writes SAS datetime values by using this ISO 8601 basic time notation:

- *hh:mm:ss*+|–*hh:mm*

*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. 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 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 000000 and 240000. If the E8601TX 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

The first example uses the local time to determine the time and the time zone offset. The second example changes the time zone to America/Adak, which is Hawaii-Aleutian Time.

| Statement | Result |
|---|---|
| `data _null_ ;`<br>`t='12:34:56't;`<br>`put t e8601tx.;`<br>`run;` | `07:34:56-05:00` |
| `options timezone='America/Adak';`<br>`data _null_ ;`<br>`t='12:34:56't;`<br>`put t e8601tx.;`<br>`run;` | `02:34:56-10:00` |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# E8601TZw.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.<fff>*+|–*hh*:*mm*.

| | |
|---|---|
| Categories: | CAS<br>Time<br>ISO 8601 |
| Alignment: | Left |
| Alias: | IS8601TZ*w.d* |
| Interaction: | When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*. |
| 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 these ISO 8601 extended time notations:

- *hh:mm:ss<.fff>+|–hh:mm*

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

- *hh:mm:ssZ*

  **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.

*.fff*
are optional fractional seconds.

+|–*hh:mm*
is an hour and minute signed offset from zero meridian time. 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 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

```
data _null_;
   input @1 time e8601tz.;
   put time=;
   put time e8601tz.;
datalines;
   23:43:44-05:00
   23:43:44Z
run;
```

| Statement | Value of time | Result |
|---|---|---|
| put time=; | 17024 | 04:43:44+00:00 |
| put time e8601tz.; | 85424 | 23:43:44+00:00 |

## See Also

"Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 45

# EUROw.d Format

Writes numeric values with a leading euro symbol (E), a comma that separates every three digits, and a period that separates the decimal fraction.

Category: Numeric

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

## Syntax

**EURO***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 6

**Range** 1-32

**Tip** If you want the euro symbol to be part of the output, be sure to choose an adequate width.

***d***

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

**Default** 0

**Range** 0-31

**Requirement** must be less than *w*

# Comparisons

- The EURO*w.d* format is similar to the EUROX*w.d* format, but EUROX*w.d* format reverses the roles of the decimal point and the comma. This convention is common in European countries.
- The EURO*w.d* format is similar to the DOLLAR*w.d* format, except that DOLLAR*w.d* format writes a leading dollar sign instead of the euro symbol.

# Example

These examples use 1254.71 as the value of amount.

| Statements | Results |
|---|---|
| | `----+----1----+----2----+----3` |
| `put amount euro10.2;` | `E1,254.71` |
| `put amount euro5.;` | `1,255` |
| `put amount euro9.2;` | `E1,254.71` |
| `put amount euro15.3;` | `E1,254.710` |

```
data _null_;
   input x;
   put x euro10.2;
   put x euro5.;
   put x euro9.2;
   put x euro15.3;
   datalines;
1254.71
;
run;
SAS Log:
```

```
 E1,254.71
1,255
E1,254.71
     E1,254.710
```

```
   /* This code determines the default length. */
   data _null_;
      input x;
      put x euro.;
      datalines;
   1
   22
   333
   4444
   55555
   666666
   7777777
   88888888
   999999999
   1234561234
   ;run;
   SAS Log:
        datalines;
       E1
      E22
     E333
   E4,444
   55,555
   666666
   7.78E6
   8.89E7
      1E9
   1.23E9
   NOTE: At least one W.D format was too small for the number to be
   printed.
   The decimal may be shifted by the "BEST" format.
```

```
/* This code determines the range. */
data _null_;
   input x;
   put x euro5.;
   put x euro6.;
   put x euro7.;
   put x euro8.;
   put x euro9.;
   put x euro9.2;
   put x euro10.;
   put x euro10.2;
   put x euro10.4;
   put x euro11.;
   put x euro11.3;
   put x euro12.;
   put x euro12.2;
   put x euro13.;
   put x euro13.2;
   datalines;
333
```

```
4444
55555
666666
7777777
88888888
999999999
1234561234
;run;
```

## See Also

**Format:**

- "EUROXw.d" on page 213

**Informats:**

- "EUROw.d" in *SAS National Language Support (NLS): Reference Guide*
- "EUROXw.d" in *SAS National Language Support (NLS): Reference Guide*

# EUROXw.d Format

Writes numeric values with a leading euro symbol (E), a period that separates every three digits, and a comma that separates the decimal fraction.

| | |
|---|---|
| Category: | Numeric |
| Alignment: | Right |
| Restriction: | This format is not supported in a DATA step that runs in CAS. |

## Syntax

**EUROX***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 6

**Range** 1-32

**Tip** If you want the euro symbol to be part of the output, be sure to choose an adequate width.

***d***

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

**Default** 0

**Range** 0-31

**Requirement** must be less than *w*

# Comparisons

- The EUROX*w.d* format is similar to the EURO*w.d* format, but EURO*w.d* format reverses the roles of the comma and the decimal point. This convention is common in English–speaking countries.
- The EUROX*w.d* format is similar to the DOLLARX*w.d* format, except that DOLLARX*w.d* format writes a leading dollar sign instead of the euro symbol.

# Example

These examples use 1254.71 as the value of amount.

| Statements | Results |
|---|---|
| | `----+----1----+----2----+----3` |
| `put amount eurox10.2;` | `E1.254,71` |
| `put amount eurox5.;` | `1.255` |
| `put amount eurox9.2;` | `E1.254,71` |
| `put amount eurox15.3;` | `E1.254,710` |

```
data _null_;
   input x;
   put x eurox10.2;
   put x eurox5.;
   put x eurox9.2;
   put x eurox15.3;
   datalines;
1254.71
; run;
SAS Log:
 E1.254,71
1.255
E1.254,71
     E1.254,710
```

```
/* This code determines the default length. */
data _null_;
   input x;
   put x eurox.;
   datalines;
1
22
333
4444
55555
666666
7777777
88888888
999999999
1234561234
;run;
SAS Log:
    E1
   E22
  E333
E4.444
55.555
666666
7.78E6
8.89E7
   1E9
1.23E9
```

**Note:** At least one W.D format was too small for the number to be printed. The decimal might be shifted by the "BEST" format.

## See Also

**Format:**

- “EUROw.d” on page 210

**Informats:**

- “EUROw.d” in *SAS National Language Support (NLS): Reference Guide*
- “EUROXw.d” in *SAS National Language Support (NLS): Reference Guide*

# Fw.d Format

Rounds the value to the nearest number that fits in the output field.

Categories: CAS
Numeric

Alignment: Right

# Syntax

**F***w*.<*d*>

## Required Argument

***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.

## Optional Argument

***d***

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional. If *d* is 0 or you omit *d*, F*w*.*d* writes the value without a decimal point.

**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

Only decimal separator and negative symbol alignment complies with Common Locale Data Repository (CLDR).

# Example

```
data one;
   x=23451.23 ;
   put x F8.1;
run;
```

SAS writes the following results to the log:

```
23451.2
```

# FLOATw.d Format

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

Categories: CAS
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. are typically 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.

Delimiters are not supported in the PUT statement's LIST output.

# 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

```
data one;
   x=1;
   y=put(x,float4.);
   put y;
   put y hex.;
run;
```

The result of the `put y=;` statement shows the character representation of the underlying hexadecimal values. The result of the `put y=hex.;` statement shows the hexadecimal value that the FLOAT format produced. The result is a hexadecimal representation of a binary number that is stored in little-endian form. Big-endian IEEE systems store the hexadecimal representation as 3F800000.

| Statement | Value of y |
|---|---|
| `put y=;` | €? |
| `put y=hex.;` | 0000803F |

# FRACTw. Format

Converts numeric values to fractions.

Categories: CAS
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 (for example, 1/2 instead of 50/100).

## Example

```
data one;
   input x;
   datalines;
0.6666666667
0.2784
;
run;
   data two;
   set one;
   put x fract8.;
run;
```

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

# HEXw. Format

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

Categories: CAS
Numeric

Alignment: Left

See: "HEXw.: Windows" in *SAS Companion for Windows*
"HEX Format: UNIX" in *SAS Companion for UNIX Environments*

## 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

```
data one;
   input x;
   datalines;
35.4
88
2.33
-150
;
run;

data two;
   set one;
   put x hex8.;
run;
```

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

# HHMMw.d Format

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

Categories: CAS
Time

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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 0 for a single-hour digit.

# Example

```
data a;
   mytime='12:59:56't;
   put mytime hhmm.;
   put mytime hhmm8.2;
run;
```

In the first example, SAS rounds up the time value 4 seconds based on the value of seconds in the SAS time value. In the second example, adding a decimal specification of 2 to the format shows that 56 seconds is 93% of a minute.

| SAS Statement | Result |
|---|---|
| `put mytime hhmm.;` | `13:00` |
| `put mytime hhmm8.2;` | `12:59.93` |

# See Also

**Formats:**


- "HOURw.d" on page 224
- "MMSSw.d" on page 240
- "TIMEw.d" on page 465
- "TODw.d" on page 470


**Functions:**

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

**Informats:**


- "TIMEw." on page 702

# HOURw.d Format

Writes time values as hours and decimal fractions of hours.

Categories: CAS
Time

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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

```
data a;
   mytime='12:59:56't;
```

```
    put mytime hour.;
run;
```

| SAS Statement | Result |
|---|---|
| put mytime hour; | 13 |

## See Also

**Formats:**

- “HHMMw.d” on page 221
- “MMSSw.d” on page 240
- “TIMEw.d” on page 465
- “TODw.d” on page 470

**Functions:**

- “HMS” in *SAS Functions and CALL Routines: Reference*
- “HOUR” in *SAS Functions and CALL Routines: Reference*
- “MINUTE” in *SAS Functions and CALL Routines: Reference*
- “SECOND” in *SAS Functions and CALL Routines: Reference*
- “TIME” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “TIMEw.” on page 702

# IBw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

See: “IB Format: UNIX” in *SAS Companion for UNIX Environments*
“IBw.d: 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 37 .

# 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 "Writing Data Generated on Big Endian and Little Endian Platforms" on page 37.

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

## Example

```
data _null_;
   x=128;
   y=put(x,ib4.);
   put y $hex8.;
run;
```

| Value of x | Result on Big Endian Platforms [1] | Result on Little Endian Platforms [1] |
|---|---|---|
| 128 | 00000080 | 80000000 |

1 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" on page 227

# IBRw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 37 .

# 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 "Writing Data Generated on Big Endian and Little Endian Platforms" on page 37.

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

# Example

```
 data _null_;
   x=128;
   y=put(x,ibr4.);
   put y $hex8.;
run;
```

| Value of x | Result[1] |
|---|---|
| 128 | 80000000 |

1 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" on page 225

# IEEEw.d Format

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

Categories: CAS
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 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** 1–8

**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 1 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

```
data _null_;
   x=1;
   test1=put(x,ieee4.);
   put test1 $hex8.;
   test2=put(x,ieee5.);
   put test2 $hex10.;
 run;
```

| Value of x | Result[1] |
|---|---|
| 1 | 3F800000 |
| | 3FF0000000 |

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

# JULDAYw. Format

Writes date values as the Julian day of the year.

Category: Date

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

See: “Julian Date Formats and Astronomical Dates” on page 10

# 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*. In this syntax, the letter d is defined as follows:

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

# Example

```
data one;
input date :mmddyy10.;
datalines;
01/12/2018
07/04/2018
;
run;

data two;
set one;
put date julday3.;
run;
```

The example table uses the input values of 21196, which is the SAS date value that corresponds to January 12, 2018, and 21369, which is the SAS date value that corresponds to July 4, 2018.

| Input Value | SAS Statement | Result |
|---|---|---|
| | | `----+----1` |
| 21196 | `put date julday3.;` | `12` |
| 21369 | `put date julday3.;` | `185` |

# JULIANw. Format

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

Categories: CAS
Date

Alignment: Left

See: "Julian Date Formats and Astronomical Dates" on page 10

## 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

```
data one;
input date :mmddyy10.;
datalines;
07/04/2018
;
run;
```

```
data two;
set one;
put date julian5.;
put date julian7.;
run;
```

The example table uses the input value of 21369, which is the SAS date value that corresponds to July 4, 2018 (the 185th day of the year).

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put date julian5.;` | `18185` |
| `put date julian7.;` | `2018185` |

## See Also

**Functions:**

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

**Informats:**

- "JULIANw." on page 645

# MDYAMPMw.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.

Categories: CAS
Date and Time

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

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

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

**Default** 0

**Range** 0–39

# 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

```
data one;
   mydt='04jul2018 14:58:23'dt;
   put mydt mdyampm25.;
run;
```

| SAS Statement | Result |
|---|---|
| `put mydt mdyampm25.` | `7/4/2018  2:58 PM` |

## See Also

**Formats:**

- "DATETIMEw.d" on page 172

**Informats:**

- "MDYAMPMw.d" on page 647

# MMDDYYw. 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 two or four digits.

Categories: CAS
Date

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 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 these forms:

*mmdd<yy>yy*

*mm/dd/<yy>yy*:

In this syntax, the letters m, d, and y and the forward slash are defined as follows:

*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

```
data one;
   mydt='02may2018'd;
   put mydt mmddyy.;
   put mydt mmddyy2.;
   put mydt mmddyy3.;
   put mydt mmddyy4.;
   put mydt mmddyy5.;
   put mydt mmddyy6.;
   put mydt mmddyy7.;
   put mydt mmddyy8.;
   put mydt mmddyy10.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+` |
| `put mydt=02may2018'd;` | `05/02/18` |
| `put mydt mmddyy2.;` | `05` |
| `put mydt mmddyy3.;` | ` 05` |
| `put mydt mmddyy4.;` | `0502` |

| SAS Statement | Result |
|---|---|
| `put mydt mmddyy5.;` | `05/02` |
| `put mydt mmddyy6.;` | `050218` |
| `put mydt mmddyy7.;` | ` 050218` |
| `put mydt mmddyy8.;` | `05/02/18` |
| `put mydt mmddyy10.;` | `05/02/2018` |

## See Also

**Formats:**

- "DATEw." on page 169
- "DDMMYYw." on page 175
- "MMDDYYxw." on page 237
- "YYMMDDw." on page 498

**Functions:**

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

**Informats:**

- "DATEw." on page 617
- "DDMMYYw." on page 621
- "YYMMDDw." on page 720

# MMDDYYxw. 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 that 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 two or four digits.

Categories: CAS
Date

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** 7

**Range** 5–32

**Interactions** When *w* has a value 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 4–32.

# Details

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

*mmdd<yy>yy*

*mmxddx<yy>yy*

In this syntax, the letters m, x, d, and y are defined as follows:

*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

```
data one;
   mydt='02may2018'd;
   put mydt mmddyyc5.;
   put mydt mmddyyd8.;
   put mydt mmddyyp10.;
   put mydt mmddyyn8.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+` |
| `put mydt mmddyyc5.;` | `05:02` |
| `put mydt mmddyyd8.;` | `05-02-18` |
| `put mydt mmddyyp10.;` | `05.02.2018` |
| `put mydt mmddyyn8.;` | `05022018` |

# See Also

**Formats:**

- “DATEw.” on page 169
- “DDMMYYxw.” on page 177
- “MMDDYYw.” on page 235
- “YYMMDDxw.” on page 501

**Functions:**

- “DAY” in *SAS Functions and CALL Routines: Reference*
- “MDY” in *SAS Functions and CALL Routines: Reference*
- “MONTH” in *SAS Functions and CALL Routines: Reference*
- “YEAR” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “MMDDYYw.” on page 649

# MMSSw.d Format

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

Categories: CAS
Time

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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

**Requirement** *d* must be less than *w*.

## Example

```
data a;
   x='8:30:21't;
   y='25:42:08't;
   put x mmss8.;
   put y mmss10.2;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put x mmss8.;` | `510:21` |
| `put y mmss10.2;` | `1542:08.00` |

## See Also

**Formats:**

- "HHMMw.d" on page 221
- "TIMEw.d" on page 465

**Functions:**

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

**Informats:**

- "TIMEw." on page 702

# MMYYw. Format

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

Categories: CAS
Date

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*. In this syntax, the letters m, M, and y are defined as follows:

*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

```
data one;
   dte='15may2018'd;
   put dte mmyy.;
   put dte mmyy5.;
   put dte mmyy6.;
   put dte mmyy7.;
   put dte mmyy10.;
run;
```

| SAS Statement | Result |
|---|---|
| | ----+----1----+ |
| put dte mmyy. | 05M2018 |
| put dte mmyy5.; | 05M18 |
| put dte mmyy6.; | 05M18 |
| put date mmyy7.; | 05M2018 |
| put date mmyy10.; | 05M2018 |

## See Also

**Formats:**


- "MMYYxw." on page 243
- "YYMMw." on page 497


# MMYYxw. 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. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits.

| | |
|---|---|
| Categories: | CAS<br>Date |
| 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 these forms:

*mm<yy>yy*

*mmx<yy>yy*

In this syntax, the letters m, x, and y are defined as follows:

*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

```
data one;
   dte='15may2018'd;
   put dte mmyyc5.;
   put dte mmyyd.;
   put dte mmyyn4.;
   put dte mmyyp8.;
   put dte mmyys10.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+` |
| `put dte mmyyc5.;` | `05:18` |
| `put dte mmyyd.;` | `05-2018` |
| `put dte mmyyn4.;` | `0518` |

| SAS Statement | Result |
|---|---|
| `put dte mmyyp8.;` | `05.2018` |
| `put dte mmyys10.;` | `05/2018` |

## See Also

**Formats:**

- “MMYYw.” on page 241
- “YYMMxw.” on page 503

# MONNAMEw. Format

Writes date values as the name of the month.

Categories: CAS
Date

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

```
data one;
   dte='15may2018'd;
   put dte monname.;
   put dte monname1.;
   put dte monname3.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put dte monname.;` | `May` |
| `put dte monname1.;` | `M` |
| `put dte monname3.;` | `May` |

## See Also

**Formats:**

- “MONTHw.” on page 246

# MONTHw. Format

Writes date values as the month of the year.

Categories: CAS
Date

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

```
data one;
   dte='15may2021'd;
   put dte month.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put dte month.;` | `5` |

## See Also

**Formats:**

- "MONNAMEw." on page 245

# MONYYw. Format

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

Categories: CAS
Date

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*. In this syntax, the letters m and y are defined as follows:

*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

```
data one;
   dte='15may2021'd;
   put dte monyy5.;
   put dte monyy7.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put dte monyy5.;` | `MAY18` |
| `put dte monyy7.;` | `MAY2021` |

## See Also

**Formats:**

- “DDMMYYw.” on page 175
- “DTMONYYw.” on page 186
- “MMDDYYw.” on page 235
- “NLDATEMw.”
- “YYMMDDw.” on page 498

**Functions:**

- “MONTH” in *SAS Functions and CALL Routines: Reference*
- “YEAR” in *SAS Functions and CALL Routines: Reference*

**Informat:**

- “MONYYw.” on page 651

# NEGPARENw.d Format

Writes negative numeric values in parentheses.

| | |
|---|---|
| Categories: | CAS<br>Numeric |
| Alignment: | Right |
| Interaction: | When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV=” in *SAS System Options: Reference*. |

## 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

```
data one;
   input x;
   datalines;
100
1000
-200
-2000
;
run;

data two;
   set one;
   put x negparen8.;
run;
```

| Value of x | Result |
|---|---|
| | `----+----1----+` |
| 100 | `  100` |
| 1000 | `1,000` |
| -200 | `(200)` |

| Value of x | Result |
| --- | --- |
| -2000 | (2,000) |

# NENGOw. Format

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

Category: Date

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## Syntax

**NENGO***w.*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 10

**Range** 2–10

## Details

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

*e*

is the first letter of the name of the imperial era (Meiji, Taisho, Showa, Heisei, or Reiwa).

*yy*

is an integer that represents the year.

*mm*

is an integer that represents the month.

*dd*

is an integer that represents the day of the month.

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

## Example

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

```
data _null_;
   date=15342;
   put date nengo3.;
   put date nengo6.;
   put date nengo8.;
   put date nengo9.;
   put date nengo10.;
run;
```

```
H14
H14/01
H.140102
H14/01/02
H.14/01/02
```

## See Also

**Informat:**

- "NENGOw." in *SAS National Language Support (NLS): Reference Guide*

# NLBESTw. Format

Writes the best numerical notation based on the locale.

Categories: CAS
Numeric

Alignment: Right

## Syntax

**NLBEST***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, use a field width of at least 8.

# Details

The NLBEST format writes the best numerical value based on the locale's decimal point and the sign mark's location. NLBEST is similar to the BEST format. For more information, see the BEST format in the *SAS Formats and Informats: Reference*.

# Example

The following code produces results based on the locale:

```
options locale=English_UnitedStates;
data test;
   x=-1257000;
   put x nlbest6.;
   put x nlbest3.;
   put "=====";
   x=-0.1;
   put x nlbest6.;
   put x nlbest3.;
   put "=====";
   x=0.1;
   put x nlbest6.;
   put x nlbest3.;
   put "=====";
   x=1257000;
   put x nlbest6.;
   put x nlbest3.;
run;
```

```
-126E4
***
=====
  -0.1
-.1
=====
   0.1
0.1
=====
1.26E6
1E6
```

```
options locale=German_Germany;
data test;
   x=-1257000;
   put x nlbest6.;
   put x nlbest3.;
```

```
   put "=====";
   x=-0.1;
   put x nlbest6.;
   put x nlbest3.;
   put "=====";
   x=0.1;
   put x nlbest6.;
   put x nlbest3.;
   put "=====";
   x=1257000;
   put x nlbest6.;
   put x nlbest3.;
run;
```

```
-126E4
***
=====
  -0,1
-,1
=====
   0,1
0,1
=====
1,26E6
1E6
```

```
options locale=ar_BH;
data test;
   x=-1257000;
   put x nlbest6.;
   put x nlbest3.;
   put "=====";
   x=-0.1;
   put x nlbest6.;
   put x nlbest3.;
   put "=====";
   x=0.1;
   put x nlbest6.;
   put x nlbest3.;
   put "=====";
   x=1257000;
   put x nlbest6.;
   put x nlbest3.;
run;
```

```
-126E4
***
=====
  -0.1
-.1
=====
   0.1
0.1
=====
1.26E6
1E6
```

# NLDATEw. Format

Writes a SAS date value as a date that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATE***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 20

**Range** 10–200

## Comparisons

NLDATE*w*. is similar to DATE*w*. and WORDDATE*w*., except that NLDATE*w*. is locale-specific.

## Example

These examples use the input value of 15760, which is the SAS date value that corresponds to February 24, 2003. The first example specifies the english_United States locale. The second example specifies the german_Germany locale.

```
options locale=English_UnitedStates;
data test;
   day=15760;
   put day nldate.;
run;
```

```
February 24, 2003
```

```
options locale=German_Germany;
data test;
```

```
   day=15760;
   put day nldate.;
run;
```

```
24. Februar 2003
```

## See Also

**Formats:**

- “NLDATEMNw.” on page 262
- “NLDATEWw.” on page 265
- “NLDATEWNw.” on page 267

# NLDATELw. Format

Writes a SAS date value as a date in the form month, date, year that is appropriate for the current SAS locale.

| | |
|---|---|
| Categories: | CAS<br>Date |
| Alignment: | Left |

## Syntax

**NLDATEL***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 18

**Range** 2–200

## Details

NLDATEL writes the date in a long-uniform pattern with the full length of the month and week names.

## Example

This example uses the date November 19, 2012.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put "+--- NLDATEL min=2 default=18
   max=200 ---+";
   put dy nldatel.;
   put dy nldatel10.;
   put dy nldatel12.;
   put dy nldatel18.;
   put dy nldatel200.;
run;
```

```
+--- NLDATEL min=2 default=18max=200 ---+
November 19, 2012
11/19/2012
Nov 19, 2012
November 19, 2012
November 19, 2012
```

# NLDATEMw. Format

Writes a SAS date value as a date in a medium-uniform pattern that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEM***w*

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 14

**Range** 2–200

## Details

NLDATEM writes the date in a medium-uniform pattern such as Nov 19, 2012.

## Example

This example specifies the date Nov 19, 2012.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put '+--- NLDATEM min=2 default=14
   max=200 ---+';
   put dy nldatem.;
   put dy nldatem8.;
   put dy nldatem14.;
   put dy nldatem200.;
run;
```

```
+--- NLDATEM min=2 default=14 max=200 ---+
Nov 19, 2012
11/19/12
Nov 19, 2012
Nov 19, 2012
```

# NLDATEMDw. Format

Writes the SAS date value as the name of the month and the day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEMD***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 16

**Range** 6-200

## Example

This example specifies the english_United States locale option.

```
option locale=en_US;
data _null_;
   dy = date();
   put dy nldatemd.;
run;
```

```
April 03
```

## See Also

**Format:**

- "NLDATEYMw." on page 268

# NLDATEMDLw. Format

Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEMDL***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 12

**Range** 5–200

## Details

NLDATEMDL writes the date in a long-uniform pattern with the full length of the month and the day such as November 19.

## Example

The following example specifies the date of November 19, 2012.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put '+--- NLDATEMDL min=5 default=12
   max=200 ---+';
   put dy nldatemdl.;
   put dy nldatemdl5.;
   put dy nldatemdl9.;
   put dy nldatemdl12.;
   put dy nldatemdl200.;
run;
```

```
+--- NLDATEMDL min=5 default=12max=200 ---+
November 19
11/19
Nov 19
November 19
November 19
```

# NLDATEMDMw. Format

Writes a SAS date value as the month and day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEMDM***w*.

## Syntax Description

***w***
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 9

**Range** 5–200

## Details

NLDATEMDM writes the date in a medium-uniform pattern with abbreviation of the month and the day using numbers and delimiters, such as Nov 19.

## Example

The following example specifies the date of November 19, 2012.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put '+--- NLDATEMDL min=5 default=9
   max=200 ---+';
   put dy nldatemdm.;
   put dy nldatemdm5.;
   put dy nldatemdm9.;
   put dy nldatemdm200.;
run;
```

```
+--- NLDATEMDL min=5 default=9max=200 ---+
Nov 19
11/19
Nov 19
Nov 19
```

# NLDATEMDSw. Format

Writes a SAS date value as the month and day of the month in a short-uniform pattern that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEMDS***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 5

**Range** 5–200

# Details

NLDATEMDS writes the date in a short-uniform pattern in full length of the month and the day using numbers and delimiters, such as MM/dd.

# Example

The following example specifies the date of November 19, 2012.

```
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put '+--- NLDATEMDL min=5 default=5
   max=200 ---+';
   put dy nldatemds.;
   put dy nldatemds5.;
   put dy nldatemds5.;
   put dy nldatemds200.;
run;
```

```
+--- NLDATEMDL min=5 default=5max=200 ---+
11/19
11/19
11/19
11/19
```

# NLDATEMNw. Format

Writes a SAS date value as the name of the month that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

# Syntax

**NLDATEMN***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the name of the month to fit the format width.

**Default** 9

**Range** 4–200

# Details

NLDATEMN writes values in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

This example demonstrates uniform pattern. This example uses a UTF-8 session and the width value is 6 for both statements.

```
option locale = zh_cn;
data test;
   x = '08jun91'd;
   put x nldatemn6.;
    = '08sep91'd;
   put x nldatemn6.;
run;
```

```
6
12
```

# Comparisons

NLDATEMN*w*. is similar to MONNAME*w*., except that NLDATEMN*w*. is locale-specific.

# Example

These examples specify the input value of 15760, which is the SAS date value that corresponds to February 24, 2003. The first example specifies the en_US locale. The second example specifies the german_Germany locale.

```
option locale=en_US;
data _null_;
   month=15760;
```

```
   put month nldatemn.;
run;
```

```
February
```

```
option locale=german_germany;
data _null_;
   month=15760;
   put month nldatemn.;
run;
```

```
Februar
```

## See Also

**Formats:**

- “NLDATEw.” on page 255
- “NLDATEWw.” on page 265
- “NLDATEWNw.” on page 267

# NLDATESw. Format

Writes a SAS date value as a date string that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATES***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 10

**Range** 2–200

## Details

NLDATES writes the date in a short-uniform pattern that contains only numbers and delimiters, such as mm/dd/yyyy.

## Example

This example specifies the date November 19, 2012. This example specifies the en_US locale.

```
option locale=en_US;
data _null_;
   dt = dt='19Nov2012:00:00:00'dt;
   dy='19Nov2012'd;
   put '+--- NLDATEMDL min=2 default=10
   max=200 ---+';
   put dy nldates.;
   put dy nldates8.;
   put dy nldates10.;
   put dy nldates200.;;
run;
```

```
+--- NLDATEMDL min=2 default=10max=200 ---+
11/19/2012
11/19/12
11/19/2012
11/19/2012
```

# NLDATEWw. Format

Writes a SAS date value as the date and the day of the week that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEW***w*.

## Syntax Description

***w***
specifies the width of the output field. If necessary, SAS abbreviates the date and the day of the week to fit the format width.

**Default** 29

**Range** 10–200

# Details

The NLDATEW format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

NLDATEW writes the date value in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

# Comparisons

NLDATEW*w*. is similar to WEEKDATE*w*., except that NLDATEW*w*. is locale specific.

# Example

These examples use the input value of 15760, which is the SAS date value that corresponds to February 24, 2003. The first example specifies the en_US locale. The second example specifies the de_DE locale.

```
options locale=en_US;
data _null_;
dy=15760;
put dy nldatew.;
put dy nldatew20.;
put dy nldatew200.;
run;
```

```
Monday, February 24, 2003
Mon, Feb 24, 2003
Monday, February 24, 2003
```

```
options locale=de_DE;
data _null_;
   dy=15760;
   put dy nldatew.;
   put dy nldatew20.;
   put dy nldatew200.;
run;
```

```
Mo., 24. Feb 2003
Mo., 24. Feb 2003
Montag, 24. Februar 2003
```

## See Also

**Formats:**

- “NLDATEw.” on page 255
- “NLDATEMNw.” on page 262
- “NLDATEWNw.” on page 267

# NLDATEWNw. Format

Writes the SAS date value as the day of the week that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEWN***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the day of the week to fit the format width.

**Default** 9

**Range** 4–200

## Details

NLDATEWN writes the date value in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

## Comparisons

NLDATEWN*w*. is similar to DOWNAME*w*., except that NLDATEWN*w*. is locale-specific.

## Example

These examples use the input value of 15760, which is the SAS date value that corresponds to February 24, 2003. The first example specifies the en_US locale. The second example specifies the de_DE locale.

```
options locale=en_US;
data _null_;
   dy=15760;
   put dy nldatewn.;
run;
```

```
Monday
```

```
options locale=de_DE;
data _null_;
   dy=15760;
   put dy nldatewn10.;
run;
```

```
Montag
```

## See Also

**Formats:**

- “NLDATEw.” on page 255
- “NLDATEMNw.” on page 262
- “NLDATEWw.” on page 265

# NLDATEYMw. Format

Writes the SAS date value as the year and the name of the month that is appropriate for the current SAS locale.

| | |
|---|---|
| Categories: | CAS<br>Date |
| Alignment: | Left |

# Syntax

**NLDATEYM***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 16

**Range** 6–200

# Details

If you specify a width of 6, but your data is larger than 6, your output contains asterisks: `d=******`. To remove the asterisks, you can use PROC LOCALEDATA. The following example uses PROC LOCALEDATA to write the date without the asterisks:

```
PROC LOCALEDATA;
   LOAD SASLOCALE;
   MODIFY key=DATE_YYMM_SHORT_FORMAT value='%b %y' ;
   SAVE REGISTRY / _ALL_ syntax=SAS;

data _null_;
   format d nldateym6.;
   d = '17OCT14'd;
   put d=;
run;
```

```
d=Oct 14
```

# Example

This example specifies the spanish_Spain locale option. This example specifies the date of April 4, 2019.

```
options locale=spanish_Spain;
data _null_;
   dy=today();
   put dy nldateym.;
   put dy nldateym12.;
   put dy nldateym200.;
run;
```

```
abril de 2019
abr de 2019
abril de 2019
```

## See Also

**Format:**

- "NLDATEMDw." on page 258

# NLDATEYMLw. Format

Writes a SAS date value as the month and year that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEYML***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 14

**Range** 5–200

## Details

NLDATEYML writes the date in a long-uniform pattern with abbreviations for the month and year, such as April 2019.

## Example

This example specifies the date April 4, 2019. This example specifies the en_US locale.

```
options locale=en_US;
data _null_;
   dt = datetime();
   dy = date();
   put "+--- NLDATEYML min=5 default=14 max=200 ---+";
   put dy nldateyml.;
   put dy nldateyml7.;
   put dy nldateyml11.;
   put dy nldateyml14.;
   put dy nldateyml200.;
run;
```

```
+--- NLDATEYML min=5 default=14 max=200 ---+
April 2019
04/2019
Apr 2019
April 2019
April 2019
```

# NLDATEYMMw. Format

Writes a SAS date value as the month and year with abbreviations that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEYMM***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 11

**Range** 5–200

## Details

NLDATEYMS writes the date in a medium-uniform pattern with abbreviations for the month and year, such as Apr 2019.

## Example

This example specifies the date April 4, 2019. This example specifies the en_US locale.

```
options locale=en_US;
data _null_;
   dt = datetime();
   dy = date();
   put "+--- NLDATEYML min=5 default=14 max=200 ---+";
   put dy nldateymm.;
   put dy nldateymm7.;
   put dy nldateymm11.;
   put dy nldateymm200.;
run;
```

```
+--- NLDATEYML min=5 default=14 max=200 ---+
Apr 2019
04/2019
Apr 2019
Apr 2019
```

# NLDATEYMSw. Format

Writes a SAS date value as a date and year that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEYMS***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 7

**Range** 5–200

## Details

NLDATEYMS writes the date in a short-uniform pattern with numbers and delimiters such as mm/yyy.

## Example

This example specifies the date April 4, 2019. This example specifies the en_US locale.

```
options locale=en_US;
data _null_;
   dt = datetime();
   dy = date();
   put "+--- NLDATEYMS min=5 default=14 max=200 ---+";
   put dy nldateyms.;
   put dy nldateyms5.;
   put dy nldateyms7.;
   put dy nldateyms200.;
run;
```

```
+--- NLDATEYMS min=5 default=14 max=200 ---+
04/2019
04/19
04/2019
04/2019
```

# NLDATEYQw. Format

Writes the SAS date value as the year and the quarter that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEYQ***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 16

**Range** 4–200

# Details

The NLDATEYQ format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

# Example

This example specifies the date April 4, 2019. This example specifies the fr_FR locale option.

```
options locale=fr_FR;
data _null_;
   dy=today();
   dt=datetime();
   put "+--- NLDATEYQ min=4 default=16 max=200 ---+";
   put ' 16' +5 dy nldateyq.;
   put ' 4' +5 dy nldateyq4.;
   put ' 14' +5 dy nldateyq14.;
   put ' 32' +5 dy nldateyq32.;
   put '200' +5 dy nldateyq200.;
run;
```

```
+--- NLDATEYQ min=4 default=16 max=200 ---+
 16     T2 2019
 4     ****
 14     T2 2019
 32     2e trimestre 2019
200
2e trimestre 2019
```

# NLDATEYQLw. Format

Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale.

Categories: CAS

Date

Alignment: Left

# Syntax

**NLDATEYQL***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 18

**Range** 4–200

# Details

NLDATEYQL writes the date in a long-uniform pattern with full length for the year and year's quarter value, such as 2nd quarter 2019.

# Example

This example specifies the date April 4, 2019. This example specifies the en_US locale option.

```
options locale=en_US;
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATEYQL min=4 default=18 max=200 ---+';
   put dy nldateyql.;
   put dy nldateyql6.;
   put dy nldateyql7.;
   put dy nldateyql18.;
   put dy nldateyql200.;
run;
```

```
+--- NLDATEYQL min=4 default=18 max=200 ---+
2nd quarter 2019
2019/2
Q2 2019
2nd quarter 2019
2nd quarter 2019
```

# NLDATEYQMw. Format

Writes a SAS date value as the year and the year's quarter value (Q1–Q4) using abbreviations that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEYQM***w*.

### Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 7

**Range** 4–200

## Details

NLDATEYQM writes the date in a medium-uniform pattern with abbreviations for the year and year's quarter value, such as Q2 2019.

## Example

The following example uses the date April 4, 2019.

```
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATEYQM min=4 default=7 max=200 ---+';
   put dy nldateyqm.;
   put dy nldateyqm6.;
   put dy nldateyqm7.;
   put dy nldateyqm200.;
run;
```

```
+--- NLDATEYQM min=4 default=7 max=200 ---+
Q2 2019
2019/2
Q2 2019
Q2 2019
```

# NLDATEYQSw. Format

Writes a SAS date value as the year and the year's quarter value (1–4) with numbers and delimiters that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEYQS***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 6

**Range** 4–200

## Details

NLDATEYQS writes the date in a short-uniform pattern with numbers and delimiters for the year and year's quarter value, such as 2019/2.

## Example

The following example specifies the date April 4, 2019.

```
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATEYQS min=4 default=6 max=200 ---+';
   put dy nldateyqs.;
```

```
   put dy nldateyqs4.;
   put dy nldateyqs6.;
   put dy nldateyqs200.;
run;
```

```
+--- NLDATEYQS min=4 default=6 max=200 ---+
2019/2
19/2
2019/2
2019/2
```

# NLDATEYRw. Format

Writes the SAS date value as the year that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEYR***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 16

**Range** 2–200

## Example

This example specifies the fr_FR locale option. This example specifies the date April 4, 2019.

```
options locale=fr_FR;
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATEYR min=4 default=6 max=200 ---+';
   put dy nldateyr.;
   put dy nldateyr2.;
   put dy nldateyr8.;
   put dy nldateyr200.;
```

```
run;
```

```
+--- NLDATEYR min=4 default=6 max=200 ---+
2019
19
2019
2019
```

# NLDATEYWw. Format

Writes the SAS date value as the year and the week that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATEYW***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 16

**Range** 5–200

## Example

This example specifies the date April 4, 2019. This example specifies the fr_FR locale option.

```
options locale=fr_FR;
data _null_;
   dt = datetime();
   dy = date();
   put "+--- NLDATEYW min=5 default=16 max=200 ---+";
   put ' 16' +5 dy nldateyw.;
   put ' 5' +5 dy nldateyw5.;
   put ' 8' +5 dy nldateyw8.;
   put ' 32' +5 dy nldateyw32.;
   put '200' +5 dy nldateyw200.;
run;
```

```
+--- NLDATEYW min=5 default=16 max=200 ---+
 16     Semaine 14 2019
 5     *****
 8     S14 19
 32     Semaine 14 2019
200
Semaine 14 2019
```

# NLDATMw. Format

Writes a SAS datetime value as a datetime that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATM***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the datetime value to fit the format width.

**Default** 30

**Range** 10–200

# Comparisons

The NLDATM*w*. format is similar to the DATETIME*w*. format, except that the NLDATM*w*. format is locale-specific.

# Example

These examples specify the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003. The first example specifies the en_US locale. The second example specifies the de_DE locale.

```
options locale=en_US;
data _null_;
   dy=1361709583;
```

```
   put dy nldatm.;
run;
```

```
24Feb2003:12:39:43
```

```
options locale=de_DE;
data _null_;
   dy=1361709583;
   put dy nldatm.;
run;
```

```
24. Februar 2003 12.39 Uhr
```

## See Also

**Formats:**

- “NLDATMAPw.” on page 281
- “NLDATMTMw.” on page 293
- “NLDATMWw.” on page 295

# NLDATMAPw. Format

Wries a SAS datetime value as a datetime with a.m. or p.m. that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMAP***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date-time value to fit the format width.

**Default** 32

**Range** 16–200

## Comparisons

The NLDATMAP*w*. format is similar to DATEAMPM*w*., except that the NLDATMAP*w*. format is locale-specific.

## Example

These examples specify the input value of 1361709583, which is the SAS date-time value that corresponds to 12:39:43 p.m. on February 24, 2003. The first example specifies the en_US locale. The second example specifies the es_MX locale.

```
options locale=en_US;
data _null_;
dy=1361709583;
put dy nldatmap.;
run;
```

```
February 24, 2003 12:39:43 PM
```

```
options locale=es_MX;
data _null_;
dy=1361709583;
put dy nldatmap200.;
run;
```

```
24 de febrero de 2003 12:39:43 p.m.
```

## See Also

**Formats:**

- “NLDATMw.” on page 280
- “NLDATMTMw.” on page 293
- “NLDATMWw.” on page 295

# NLDATMDTw. Format

Writes the SAS datetime value as the name of the month, day of the month and year that is appropriate for the current SAS locale.

Categories: CAS
Date
Date and Time

Alignment: Left

# Syntax

**NLDATMDT***w*.

## Syntax Description

***w***

specifies the width of the output field

**Default** 18

**Range** 10-200

# Details

NLDATMDT writes the date value in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

# Example

This example specifies the value 86400, which corresponds to January 2, 1960. The first example specifies the english_United States locale. The second example specifies the turkish_Turkey locale.

```
options locale=en_US;
data _null_;
   x=86400;
   put x nldatmdt.;
run;
```

```
January 2, 1960
```

```
options locale=tr_TR;
data _null_;
x=86400;
put x nldatmdt200.;
run;
```

```
2 Ocak 1960
```

## See Also

**Formats:**

- “NLDATMMDw.” on page 286

# NLDATMLw. Format

Writes a SAS datetime value as a long representation of the date that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATML***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 30

**Range** 9–200

## Details

NLDATML writes the date in a long-uniform pattern with the full length of the month, date, year, and time, such as April 10, 2019 03:13:27 PM.

## Example

```
data _null_;
  dt = datetime();
  dy = date();
  put "+--- NLDATML min=9 default=30
max=200 ---+";
  put dt nldatml.;
```

```
  put dt nldatml19.;
  put dt nldatml24.;
  put dt nldatml30.;
  put dt nldatml200.;
run;
```

```
+--- NLDATML min=9 default=30max=200 ---+
April 10, 2019 03:13:27 PM
04/10/2019 15:13:27
Apr 10, 2019 03:13:27 PM
April 10, 2019 03:13:27 PM
April 10, 2019 03:13:27 PM
```

# NLDATMMw. Format

Writes a SAS datetime value as a medium representation of the date that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMM***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 24

**Range** 9–200

## Details

NLDATMM writes the date in a medium-uniform pattern with abbreviations of the month and week names, such as Apr 10, 2019 03:23:34 PM.

## Example

```
data _null_;
  dt = datetime();
  dy = date();
  put '+--- NLDATMM min=9 default=30
max=200 ---+';
  put dt NLDATMM.;
  put dt NLDATMM19.;
  put dt NLDATMM24.;
  put dt NLDATMM30.;
  put dt NLDATMM200.;
run;
```

```
+--- NLDATMM min=9 default=30max=200 ---+
Apr 10, 2019 03:23:34 PM
04/10/2019 15:23:34
Apr 10, 2019 03:23:34 PM
Apr 10, 2019 03:23:34 PM
Apr 10, 2019 03:23:34 PM
```

# NLDATMMDw. Format

Writes the SAS datetime value as the name of the month and the day of the month that is appropriate for the current SAS locale.

Categories: CAS
Date
Date and Time

Alignment: Left

## Syntax

**NLDATMMD***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 16

**Range** 6–200

## Example

This example uses the en_US locale option.

```
options locale=en_US;
data _null_;
   x=put(86400,nldatmmd.);
   put x=;
run;
```

```
x=January 02
```

## See Also

**Format:**

- "NLDATMYMw." on page 299

# NLDATMMDLw. Format

Writes a SAS datetime value as the full-length of the month and day of the month that is appropriate for the current SAS locale.

| | |
|---|---|
| Categories: | CAS<br>Date |
| Alignment: | Left |

## Syntax

**NLDATMMDL***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 12

**Range** 5–200

## Details

NLDATMMDL writes the date in a long-uniform pattern with full-length of the month and the day, such as November 19.

## Example

The following example uses the date of April 12, 2019.

```
data _null_;
  dt = datetime();
  dy = date();
  put '+--- NLDATMMDL min=5 default=12
max=200 ---+';
  put dt nldatmmdl.;
  put dt nldatmmdl5.;
  put dt nldatmmdl9.;
  put dt nldatmmdl12.;
  put dt nldatmmdl200.;
run;
```

```
+--- NLDATMMDL min=5 default=12max=200 ---+
April 12
04/12
Apr 12
April 12
April 12
```

# NLDATMMDMw. Format

Writes a SAS datetime value as the month and day of the month using abbreviations that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATMMDM***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 9

**Range** 5–200

# Details

NLDATMMDM writes the date in a medium-uniform pattern with abbreviations of the month and the day, such as Nov 19.

# Example

The following example uses the date of April 12, 2019.

```
data _null_;
  dt = datetime();
  dy = date();
  put '+--- NLDATMMDM min=5 default=9
max=200 ---+';
  put dt nldatmmdm.;
  put dt nldatmmdm5.;
  put dt nldatmmdm9.;
  put dt nldatmmdm200.;
run;
```

```
+--- NLDATMMDM min=5 default=9max=200 ---+
Apr 12
04/12
Apr 12
Apr 12
```

# NLDATMMDSw. Format

Writes a SAS datetime value as the month and day of the month using numbers and delimiters that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

# Syntax

**NLDATMMDS***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 5

**Range** 5–200

# Details

NLDATMMDS writes the date in a short-uniform pattern with numbers and delimiters of the month and the day, such as 4/12.

# Example

The following example uses the date of April 12, 2019.

```
data _null_;
  dt = datetime();
  dy = date();
  put '+--- NLDATMMDS min=5 default=5 max=200 ---+';
  put dt nldatmmds.;
  put dt nldatmmds5.;
  put dt nldatmmds5.;
  put dt nldatmmds200.;
run;
```

```
+--- NLDATMMDS min=5 default=5max=200 ---+
04/12
04/12
04/12
04/12
```

# NLDATMMNw. Format

Writes the SAS datetime value as the name of the month that is appropriate for the current SAS locale.

Categories: CAS
Date

Date and Time

Alignment: Left

# Syntax

**NLDATMMN***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 9

**Range** 4–200

# Details

NLDATMMN writes values in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

# Example

This example uses the en_US locale option.

```
options locale=en_US;
data _null_;
  dt = datetime();
  dy = date();
  put '+--- NLDATMMN min=5 default=10 max=200 ---+';
  put dt nldatmmn.;
  put dt nldatmmn4.;
  put dt nldatmmn10.;
  put dt nldatmmn200.;
run;
```

```
+--- NLDATMMN min=5 default=10max=200 ---+
April
Apr
April
April
```

# NLDATMSw. Format

Writes a SAS datetime value as the short representation of the date that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMS***w*.

## Syntax Description

***w***
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 19

**Range** 9–200

## Details

NLDATMS writes the date in a short-uniform pattern with number and delimiters, such as MM/DD/YYYY hh:mm:ss.

## Example

This example uses the date April 12, 2019.

```
data _null_;
  dt = datetime();
  dy = date();
  put '+--- NLDATMS min=9 default=19 max=200---+';
  put dt nldatms.;
  put dt nldatms10.;
  put dt nldatms19.;
  put dt nldatms200.;
run;
```

```
+--- NLDATMS min=9 default=19 max=200---+
04/12/2019 11:50:53
041219 11
04/12/2019 11:50:53
04/12/2019 11:50:53
```

# NLDATMTMw. Format

Writes the time portion of a SAS datetime value as a time of day that is appropriate for the current SAS locale.

Categories: CAS
Date and Time
Time

Alignment: Left

## Syntax

**NLDATMTM***w*.

## Syntax Description

***w***
specifies the width of the output field.

**Default** 16

**Range** 16–200

## Comparisons

The NLDATMTM*w*. format is similar to the TOD*w*. format, except that the NLDATMTM*w*. format is locale-specific.

## Example

These examples use the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003.

This example specifies the English_United_States locale.

```
options locale=en_US;
data one;
```

```
   event=1361709583;
   put event nldatmtm.;
run;
```

```
12:39:43
```

This example specifies the German_Germany locale.

```
options locale=en_US;
data one;
   event=1361709583;
   put event nldatmtm.;
run;
```

```
12.39 Uhr
```

## See Also

**Formats:**

- “NLDATMw.” on page 280
- “NLDATMAPw.” on page 281
- “NLDATMWw.” on page 295

# NLDATMTZw. Format

Writes the time portion of the SAS datetime value as the time of day and time zone that is appropriate for the current SAS locale.

| | |
|---|---|
| Categories: | CAS<br>Date and Time<br>Time |
| Alignment: | Left |

## Syntax

**NLDATMTZ***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 32

**Range** 16–200

## Example

This example uses the current datetime value. This example specifies the French_France locale.

```
options locale=fr_FR;
data test;
   x=datetime();
   put x=nldatmtz.;
run;
```

```
x=10 h 08 -0400
```

# NLDATMWw. Format

Writes SAS datetime values as the day of the week and the datetime that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMW***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the day of week and datetime to fit the format width.

**Default** 41

**Range** 16–200

# Details

The NLDATMW format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths for information about recommended widths for locale and encoding combinations. You might need to use the suggested width for the NLDATMW format.

NLDATMW writes the date value in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

# Comparisons

The NLDATMW*w*. format is similar to the TWMDY*w*. format, except that the NLDATMW*w*. format is locale-specific.

# Example

This example uses the input value of 1361709583, which is the SAS datetime value that corresponds to 12:39:43 p.m. on February 24, 2003. This example specifies the English_United States locale.

```
options locale=English_UnitedStates;
data one;
   x=put(1361709583,nldatmw.);
   y=put(1361709583,nldatmw30.);
   z=put(1361709583,nldatmw200.);
   put x=;
   put y=;
   put z=;
run;
```

```
x=Monday, February 24, 2003 12:39:43 PM
y=Mon, Feb 24, 2003 12:39:43 PM
z=Monday, February 24, 2003 12:39:43 PM
```

# See Also

**Formats:**


- “NLDATMw.” on page 280
- “NLDATMAPw.” on page 281
- “NLDATMTMw.” on page 293

# NLDATMWNw. Format

Writes a SAS datetime value as the day of the week that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMWN***w*.

### Syntax Description

***w***
specifies the width of the output field.

**Default** 9

**Range** 4–200

## Details

NLDATMWN writes the date value in a uniform pattern which ensures that the output has the same pattern when using the same format width but different input data.

## Example

This example writes the SAS datetime value as a day of the week. This example specifies the English_United States locale, which is the default SAS session encoding.

```
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATMWN min=4 default=9 max=200 ---+';
   put dt nldatmwn;
   put dt nldatmwn4.;
   put dt nldatmwn9.;
   put dt nldatmwn200.;
run;
```

```
+--- NLDATMWN min=4 default=9 max=200 ---+
Tuesday
Tue
Tuesday
Tuesday
```

# NLDATMWZw. Format

Writes SAS datetime values as day-of-week, datetime, and time zone values that are appropriate.

Categories: CAS
Date
Date and Time

Alignment: Left

## Syntax

**NLDATMWZ***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the day of week and datetime to fit the format width.

**Default** 40

**Range** 16–200

## Details

The NLDATMWZ format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

## Example

This example uses the current datetime value. This example specifies the French_France locale.

```
options locale=fr_FR;
```

```
data test;
   x=datetime();
   put x=nldatmwz.;
run;
```

```
x=mardi 16 avril 2019 10:35:06 -0400
```

# NLDATMYMw. Format

Writes the SAS datetime value as the month and year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMYM***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 16

**Range** 6–200

# Example

This example uses the en_US locale option.

```
options locale=en_US;
data _null_;
   x=put(86400,nldatmym.);
   y=put(86400,nldatmym12.);
   put x=;
   put y=;
run;
```

```
x=January 1960
y=January 1960
```

## See Also

**Format:**

- "NLDATMMDw." on page 286

# NLDATMYMLw. Format

Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATMYML***w*.

### Syntax Description

***w***
specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 14

**Range** 5–200

## Details

NLDATMYML writes the date in a long-uniform pattern with full length of the month and year, such as November 2012.

## Example

The following example uses the date April 16, 2019.

```
data _null_;
   dt = datetime();
   put '+--- NLDATMYML min=5 default=14 max=200 ---+';
   put dt nldatmyml.;
   put dt nldatmyml7.;
   put dt nldatmyml11.;
```

```
   put dt nldatmyml14.;
   put dt nldatmyml200.;
run;
```

```
+--- NLDATMYML min=5 default=14max=200 ---+
April 2019
04/2019
Apr 2019
April 2019
April 2019
```

# NLDATMYMMw. Format

Writes a SAS datetime value as the month and the year that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATMYMM***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 11

**Range** 5–200

## Details

NLDATMYMM writes the date in a medium-uniform pattern with abbreviations of the month and year, such as Nov 2012.

## Example

The following example uses the date April 16, 2019.

```
data _null_;
```

```
   dt = datetime();
   dy = date();
   put '+--- NLDATMYMM min=5 default=11 max=200 ---+';
   put dt nldatmymm.;
   put dt nldatmymm7.;
   put dt nldatmymm11.;
   put dt nldatmymm200.;
run;
```

```
+--- NLDATMYMM min=5 default=11 max=200 ---+
Apr 2019
04/2019
Apr 2019
Apr 2019
```

# NLDATMYMSw. Format

Writes a SAS datetime value as the month and year with numbers and a delimiter that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

## Syntax

**NLDATMYMS***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 7

**Range** 5–200

## Details

NLDATMYMS writes the date in a short-uniform pattern with numbers and a delimiter for the month and year, such as 11/2012.

## Example

The following example uses the date April 16, 2019.

```
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATMYMS min=5 default=7 max=200 ---+';
   put dt nldatmyms.;
   put dt nldatmyms5.;
   put dt nldatmyms7.;
   put dt nldatmyms200.;
run;
```

```
+--- NLDATMYMS min=5 default=7 max=200 ---+
04/2019
04/19
04/2019
04/2019
```

# NLDATMYQw. Format

Writes the SAS datetime value as the quarter and the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMYQ***w*.

### Syntax Description

***w***
specifies the width of the output field.

**Default** 16

**Range** 4–200

## Details

The NLDATMYQ format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and

time names are too long. Please refer to Exceptions for Date and Time Default Widths for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

## Example

This example uses the en_US locale option.

```
options locale=en_US;
data _null_;
   dy=today();
   dt=datetime();
   put "+--- NLDATMYQ min=4 default=16 max=200 ---+";
   put ' 16' +5 dt nldatmyq.;
   put '  4' +5 dt nldatmyq4.;
   put ' 14' +5 dt nldatmyq14.;
   put ' 32' +5 dt nldatmyq32.;
   put '200' +5 dt nldatmyq200.;
run;
```

```
+--- NLDATMYQ min=4 default=16 max=200 ---+
 16     2nd quarter 2019
  4     ****
 14     Q2 2019
 32     2nd quarter 2019
200
2nd quarter 2019
```

# NLDATMYQLw. Format

Writes a SAS datetime value as the year's quarter value (1–4) and the year that is appropriate for the current SAS locale.

Category: Date

Alignment: Left

## Syntax

**NLDATMYQL***w*.

### Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 18

**Range** 4–200

## Details

NLDATMYQL writes the date in a long uniform pattern in full length of the year's quarter and then the year, such as 4th quarter 2012.

## Example

The following example uses the date of April 16, 2019. This example specifies the en_US locale.

```
options locale=en_US;
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATMYQL min=4 default=18 max=200 ---+';
   put dt nldatmyql.;
   put dt nldatmyql6.;
   put dt nldatmyql7.;
   put dt nldatmyql18.;
   put dt nldatmyql200.;
run;
```

```
+--- NLDATMYQL min=4 default=18 max=200 ---+
2nd quarter 2019
2019/2
Q2 2019
2nd quarter 2019
2nd quarter 2019
```

# NLDATMYQMw. Format

Writes a SAS datetime value as the year's quarter (1–4) and the year that is appropriate for the current SAS locale.

Categories: CAS
Date

Alignment: Left

# Syntax

**NLDATMYQM***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 7

**Range** 4–200

# Details

NLDATMYQM writes the date in a medium-uniform pattern of the year's quarter and then the year, such as Q2 2019.

# Example

The following example uses the date of April 16, 2019.

```
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATMYQM min=4 default=7 max=200 ---+';
   put dt nldatmyqm.;
   put dt nldatmyqm6.;
   put dt nldatmyqm7.;
   put dt nldatmyqm200.;
run;
```

```
+--- NLDATMYQM min=4 default=7 max=200 ---+
Q2 2019
2019/2
Q2 2019
Q2 2019
```

# NLDATMYQSw. Format

Writes a SAS datetime value as the year and the quarter (1-4) using numbers and a delimiter that is appropriate for the current SAS locale.

Categories: CAS

Date

Alignment: Left

# Syntax

**NLDATMYQS***w*.

## Syntax Description

***w***

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

**Default** 6

**Range** 4–200

# Details

NLDATMYQS writes the date in a short-uniform pattern of the year and year's quarter value using numbers and a delimiter, such as 2012/4.

# Example

The following example uses the date April 16, 2019.

```
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATMYQS min=4 default=6 max=200 ---+';
   put dt nldatmyqs.;
   put dt nldatmyqs4.;
   put dt nldatmyqs6.;
   put dt nldatmyqs200.;
run;
```

```
+--- NLDATMYQS min=4 default=6 max=200 ---+
2019/2
19/2
2019/2
2019/2
```

# NLDATMYRw. Format

Writes the SAS datetime value as the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMYR***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 16

**Range** 2–200

## Example

This example uses the en_US locale option, which is the default SAS session encoding.

```
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATMYR min=2 default=16 max=200 ---+';
   put dt nldatmyr.;
   put dt nldatmyr2.;
   put dt nldatmyr32.;
   put dt nldatmyr200.;
run;
```

```
+--- NLDATMYR min=2 default=16 max=200 ---+
2019
19
2019
2019
```

# NLDATMYWw. Format

Writes the SAS datetime value as the week number and the year that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMYW***w*.

### Syntax Description

***w***
specifies the width of the output field.

**Default** 16

**Range** 5–200

## Example

This example uses the en_US locale option, which is the default SAS session encoding.

```
data _null_;
   dt = datetime();
   dy = date();
   put '+--- NLDATMYW min=5 default=16 max=200 ---+';
   put ' 16' +5 dt nldatmyw.;
   put '  5' +5 dt nldatmyw5.;
   put '  8' +5 dt nldatmyw8.;
   put ' 32' +5 dt nldatmyw32.;
   put '200' +5 dt nldatmyw200.;
run;
```

```
+--- NLDATMYW min=5 default=16 max=200 ---+
 16     Week 16 2019
  5     *****
  8     W16 19
 32     Week 16 2019
200
Week 16 2019
```

# NLDATMZw. Format

Writes SAS datetime values as datetime and time zone values that are appropriate for the current SAS locale.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**NLDATMZ***w*.

### Syntax Description

***w***
specifies the width of the output field.

**Default** 40

**Range** 16–200

## Example

This example uses the current datetime value. This example specifies the fr_FR locale.

```
options locale=fr_FR;
data test;
   x=datetime();
   put x=nldatmz.;
run;
```

```
x=16 avril 2019 11 h 40 -0400
```

# NLMNIAEDw.d Format

Writes the monetary format of the international expression for the United Arab Emirates.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNIAED***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 3

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_AE;
x=put(-1234.56789,nlmniaed32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ar_AE | AED 1,234.57- |
| en_US | (AED1,234.57) |
| es_ES | -1.234,57 AED |

# See Also

**Format:**

- “NLMNLAEDx.d” on page 354

# NLMNIAUDw.d Format

Writes the monetary format of the international expression for Australia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIAUD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_AU;
x=put(-1234.56789,nlmniaud32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| en_AU | (AUD1,234.57) |
| en_US | (AUD1,234.57) |
| es_ES | -1.234,57 AUD |

## See Also

**Format:**

- "NLMNLAUDw.d" on page 356

# NLMNIBGNw.d Format

Writes the monetary format of the international expression for Bulgaria.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIBGN***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=bg_BG;
x=put(-1234.56789,nlmnibgn32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| bg_BG | -1 234,57 BGN |
| en_US | (BGN1,234.57) |
| es_ES | -1.234,57 BGN |

## See Also

**Format:**

- "NLMNLBGNw.d" on page 357

# NLMNIBRLw.d Format

Writes the monetary format of the international expression for Brazil.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIBRL***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pt_BR;
x=put(-1234.56789,nlmnibrl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| pt_BR | (BRL1.234,57) |
| en_US | (BRL1,234.57) |
| es_ES | -1.234,57 BRL |

## See Also

**Format:**

- "NLMNLBRLw.d" on page 358

# NLMNICADw.d Format

Writes the monetary format of the international expression for Canada.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNICAD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_CA;
x=put(-1234.56789,nlmnicad32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| en_CA | (CAD1,234.57) |
| en_US | (CAD1,234.57) |
| es_ES | -1.234,57 CAD |

# See Also

**Format:**

- “NLMNLCADw.d” on page 359

# NLMNICHFw.d Format

Writes the monetary format of the international expression for Liechtenstein and Switzerland.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNICHF***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=fr_CH;
x=put(-1234.56789,nlmnichf32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| fr_CH | CHF-1 234.57 |
| en_US | (CHF1,234.57) |
| es_ES | -1.234,57 CHF |

# See Also

**Format:**

- “NLMNLCHFw.d” on page 360

# NLMNICNYw.d Format

Writes the monetary format of the international expression for China.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNICNY***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 02

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_CN;
x=put(-1234.56789,nlmnicny32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| zh_CN | -CNY1,234.57 |
| en_US | (CNY1,234.57) |
| es_ES | -1.234,57 CNY |

## See Also

**Format:**

- "NLMNLCNYw.d" on page 361

# NLMNICZKw.d Format

Writes the monetary format of the international expression for the Czech Republic.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNICZK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=cs_CZ;
x=put(-1234.56789,nlmniczk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| cs_CZ | -1 234,57 CZK |
| en_US | (CZK1,234.57) |
| es_ES | -1.234,57 CZK |

## See Also

**Format:**

- "NLMNLCZKw.d" on page 363

# NLMNIDKKw.d Format

Writes the monetary format of the international expression for Denmark, Faroe Island, and Greenland.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIDKK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=da_dk;
x=put(-1234.56789,nlmnidkk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| da_DK | -1.234,57 DKK |
| en_US | (DKK1,234.57) |
| es_ES | -1.234,57 DKK |

## See Also

**Format:**

- "NLMNLDKKw.d" on page 364

# NLMNIEEKw.d Format

Writes the monetary format of the international expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIEEK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=et_EE;
x=put(-1234.56789,nlmnieek32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| et_EE | -1 234,57 EEK |
| en_US | (EEK1,234.57) |
| es_ES | -1.234,57 EEK |

# See Also

**Format:**

- “NLMNLEEKw.d” on page 365

# NLMNIEGPw.d Format

Writes the monetary format of the international expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNIEGP***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 3

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=ar_EG;
x=put(-1234.56789,nlmniegp32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ar_EG | EGP 1,234.57- |
| en_US | (EGP1,234.57) |
| es_ES | -1.234,57 EGP |

# See Also

**Format:**

- “NLMNLEGPw.d” on page 366

# NLMNIEURw.d Format

Writes the monetary format of the international expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIEUR***w.d*

### Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session..

The output value depends on the locale. You can set the locale option using this code. In the following example, the LOCALE= system option is set to de_DE.

```
option locale=de_DE;
x=put(-1234.56789,nlmnieur32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| de_DE | -1.234,57 EUR |
| en_US | (EUR1,234.57) |
| es_ES | -1.234,57 EUR |

## See Also

**Format:**

- "NLMNLEURw.d" on page 368

# NLMNIGBPw.d Format

Writes the monetary format of the international expression for the United Kingdom.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIGBP***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=en_GB;
x=put(-1234.56789,nlmnigbp32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES

| Locale | Formatted value |
|---|---|
| en_GB | -GBP1,234.57 |
| en_US | (GBP1,234.57) |
| es_ES | -1.234,57 GBP |

## See Also

**Format:**

- "NLMNLGBPw.d" on page 369

# NLMNIHKDw.d Format

Writes the monetary format of the international expression for Hong Kong.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIHKD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=en_HK;
x=put(-1234.56789,nlmnihkd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| en_HK | (HKD1,234.57) |
| en_US | (HKD1,234.57) |
| es_ES | -1.234,57 HKD |

## See Also

**Format:**

- "NLMNLHKDw.d" on page 370

# NLMNIHRKw.d Format

Writes the monetary format of the international expression for Croatia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIHRK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=hr_HR;
x=put(-1234.56789,nlmnihrk32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| hr_HR | -1.234,57 HRK |
| en_US | (HRK1,234.57) |
| es_ES | -1.234,57 HRK |

# See Also

**Format:**

- “NLMNLHRKw.d” on page 371

# NLMNIHUFw.d Format

Writes the monetary format of the international expression for Hungary.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNIHUF***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=hu_HU
x=put(-1234.56789,nlmnihuf32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| hu_HU | -1 234,57 HUF |
| en_US | (HUF1,234.57) |
| es_ES | -1.234,57 HUF |

# See Also

**Format:**

- “NLMNLHUFw.d” on page 372

# NLMNIIDRw.d Format

Writes the monetary format of the international expression for Indonesia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIIDR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=id_ID;
x=put(-1234.56789,nlmniidr32.2);
```

The output value depends on the locale. You can set the locale option using this code.

| Locale | Formatted value |
|---|---|
| id_ID | -IDR1.234,57 |
| en_US | (IDR1,234.57) |
| es_ES | -1.234,57 IDR |

## See Also

**Format:**

- "NLMNLIDRw.d" on page 374

# NLMNIILSw.d Format

Writes the monetary format of the international expression for Israel.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIILS***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=he_IL;
x=put(-1234.56789,nlmniils32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| he_IL | -1,234.57 ILS |
| en_US | (ILS1,234.57) |
| es_ES | -1.234,57 ILS |

## See Also

**Format:**

- "NLMNLILSw.d" on page 375

# NLMNIINRw.d Format

Writes the monetary format of the international expression for India.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIINR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=hi_IN;
x=put(-1234.56789,nlmniinr32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| hi_IN | -INR 1,234.57 |
| en_US | (INR1,234.57) |
| es_ES | -1.234,57 INR |

## See Also

**Format:**

- "NLMNLINRw.d" on page 376

# NLMNIJPYw.d Format

Writes the monetary format of the international expression for Japan.

Category: Numeric

Alignment: Left

## Syntax

**NLMNIJPY***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 0

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ja_JP;
x=put(-1234.56789,nlmnijpy32.2);
```

The output value depends on the locale. You can set the locale option using this code.

| Locale | Formatted value |
|---|---|
| ja_JP | -JPY1,234.57 |
| en_US | (JPY1,234.57) |
| es_ES | -1.234,57 JPY |

## See Also

**Format:**

- “NLMNLJPYw.d” on page 377

# NLMNIKRWw.d Format

Writes the monetary format of the international expression for South Korea.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIKRW***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 0

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ko_KR;
x=put(-1234.56789,nlmnikrw32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ko_KR | -KRW1,234.57 |
| en_US | (KRW1,234.57) |
| es_ES | -1.234,57 KRW |

# See Also

**Format:**

- “NLMNLKRWw.d” on page 378

# NLMNILTLw.d Format

Writes the monetary format of the international expression for Lithuania.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNILTL***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale-lt_LT;
x=put(-1234.56789,nlmniltl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| lt_LT | -1 234,57 LTL |
| en_US | (LTL1,234.57) |
| es_ES | -1.234,57 LTL |

# See Also

**Format:**

- “NLMNLLTLw.d” on page 380

# NLMNILVLw.d Format

Writes the monetary format of the international expression for Latvia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNILVL***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=lv_LV;
x=put(-1234.56789,nlmnilvl32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| lv_LV | -1 234,57 LVL |
| en_US | (LVL1,234.57) |
| es_ES | -1.234,57 LVL |

## See Also

**Format:**

- "NLMNLLVLw.d" on page 381

# NLMNIMOPw.d Format

Writes the monetary format of the international expression for Macau.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIMOP***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=zh_MO;
x=put(-1234.56789,nlmnimop32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| zh_MO | -MOP1,234.57 |
| en_US | (MOP1,234.57) |
| es_ES | -1.234,57 MOP |

## See Also

**Format:**

- "NLMNLMOPw.d" on page 382

# NLMNIMXNw.d Format

Writes the monetary format of the international expression for Mexico.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIMXN***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
option locale=es_MX;
x=put(-1234.56789,nlmnimxn32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| es_MX | -1,234.57 MXN |
| en_US | (MXN1,234.57) |
| es_ES | -1.234,57 MXN |

## See Also

**Format:**

- "NLMNLMXNw.d" on page 383

# NLMNIMYRw.d Format

Writes the monetary format of the international expression for Malaysia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIMYR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=ms_MY;
x=put(-1234.56789,nlmnimyr32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ms_MY | (MYR1,234.57) |
| en_US | (MYR1,234.57) |
| es_ES | -1.234,57 MYR |

# See Also

**Format:**

- “NLMNLMYRw.d” on page 384

# NLMNINOKw.d Format

Writes the monetary format of the international expression for Norway.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNINOK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=no_NO;
x=put(-1234.56789,nlmninok32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| no_NO | -NOK 1 234,57 |
| en_US | (NOK1,234.57) |
| es_ES | -1.234,57 NOK |

# See Also

**Format:**

- “NLMNLNOKw.d” on page 386

# NLMNINZDw.d Format

Writes the monetary format of the international expression for New Zealand.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNINZD***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_NZ;
x=put(-1234.56789,nlmninzd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| en_NZ | (NZD1,234.57) |
| en_US | (NZD1,234.57) |
| es_ES | -1.234,57 NZD |

## See Also

**Format:**

- "NLMNLNZDw.d" on page 387

# NLMNIPLNw.d Format

Writes the monetary format of the international expression for Poland.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIPLN***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pl_PL;
x=put(-1234.56789,nlmniplnl32.2;
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| pl_PL | -1 234,57 PLN |
| en_US | (PLN1,234.57) |
| es_ES | -1.234,57 PLN |

## See Also

**Format:**

- "NLMNLPLNw.d" on page 388

# NLMNIRUBw.d Format

Writes the monetary format of the international expression for Russia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIRUB***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ru_RU;
x=put(-1234.56789,nlmnirub32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ru_RU | -1 234,57 RUB |
| en_US | (RUB1,234.57) |
| es_ES | -1.234,57 RUB |

## See Also

**Format:**

- "NLMNLRUBw.d" on page 389

# NLMNISEKw.d Format

Writes the monetary format of the international expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNISEK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=sv_SE;
x=put(-1234.56789,nlmnisek32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| sv_SE | -1 234,57 SEK |
| en_US | (SEK1,234.57) |
| es_ES | -1.234,57 SEK |

# See Also

**Format:**

- “NLMNLSEKw.d” on page 390

# NLMNISGDw.d Format

Writes the monetary format of the international expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNISGD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_SG;
x=put(-1234.56789,nlmnisgd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| zh_SG | -SGD1,234.57 |
| en_US | (SGD1,234.57) |
| es_ES | -1.234,57 SGD |

# See Also

**Format:**

- “NLMNLSGDw.d” on page 392

# NLMNITHBw.d Format

Writes the monetary format of the international expression for Thailand.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNITHB***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=th_TH;
x=put(-1234.56789,nlmnithb32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| th_TH | THB-1,234.57 |
| en_US | (THB1,234.57) |
| es_ES | -1.234,57 THB |

## See Also

**Format:**

- "NLMNLTHBw.d" on page 393

# NLMNITRYw.d Format

Writes the monetary format of the international expression for Turkey.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNITRY***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=tr_TR;
x=put(-1234.56789,nlmnitry32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| tr_TR | -1.234,57 TRY |
| en_US | (TRY1,234.57) |
| es_ES | -1.234,57 TRY |

## See Also

**Format:**

- "NLMNLTRYw.d" on page 394

# NLMNITWDw.d Format

Writes the monetary format of the international expression for Taiwan.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNITWD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_TW;
x=put(-1234.56789,nlmnitwd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| zh_TW | -TWD1,234.57 |
| en_US | (TWD1,234.57) |
| es_ES | -1.234,57 TWD |

## See Also

**Format:**

- "NLMNLTWDw.d" on page 395

# NLMNIUSDw.d Format

Writes the monetary format of the international expression for Puerto Rico and the United States.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIUSD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=;
x=put(-1234.56789,nlmniusd32.2);
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| es_US | -1,234.57 USD |
| en_US | (USD1,234.57) |
| es_ES | -1.234,57 USD |

## See Also

**Format:**

- "NLMNLUSDw.d" on page 396

# NLMNIZARw.d Format

Writes the monetary format of the international expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNIZAR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=af_ZA;
x=put(-1234.56789,nlmnizar32.2;
```

This table shows the output for the international value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| af_ZA | -ZAR1 234,57 |
| en_US | (ZAR1,234.57) |
| es_ES | -1.234,57 ZAR |

# See Also

**Format:**

- “NLMNLZARw.d” on page 397

# NLMNLAEDx.d Format

Writes the monetary format of the local expression for the United Arab Emirates.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNLAED***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 3

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_AE
x=put(-1234.56789,nlmnlaed32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ar_AE | 1,234.57 د.إ- |
| en_US | (1,234.57د.إ) |
| es_ES | -1.234,57 د.إ |

# See Also

**Format:**

- “NLMNIAEDw.d” on page 310

# NLMNLAUDw.d Format

Writes the monetary format of the local expression for Australia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLAUD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_AU;
x=put(-1234.56789,nlmnlaud32.2);
```

The output value depends on the locale. You can set the locale option using this code.

| Locale | Formatted value |
|---|---|
| en_AU | (AU$1,234.57) |
| en_US | (AU$1,234.57) |
| es_ES | -1.234,57 AU$ |

## See Also

**Format:**

- "NLMNIAUDw.d" on page 312

# NLMNLBGNw.d Format

Writes the monetary format of the local expression for Bulgaria.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLBGN***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=bg_BG;
x=put(-1234.56789,nlmnlbgn32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| bg_BG | -1 234,57 лв |
| en_US | (лв1,234.57) |
| es_ES | -1.234,57 лв |

## See Also

**Format:**

- “NLMNIBGNw.d” on page 313

# NLMNLBRLw.d Format

Writes the monetary format of the local expression for Brazil.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLBRL***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pt_BR;
x=put(-1234.56789,nlmnlbrl32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| pt_BR | (R$1.234,57) |
| en_US | (R$1,234.57) |
| es_ES | -1.234,57 R$ |

## See Also

**Format:**

- "NLMNIBRLw.d" on page 314

# NLMNLCADw.d Format

Writes the monetary format of the local expression for Canada.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLCAD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_CA;
x=put(-1234.56789,nlmnlcad32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| en_CA | (CA$1,234.57) |
| en_US | (CA$1,234.57) |
| es_ES | -1.234,57 CA$ |

## See Also

**Format:**

- "NLMNICADw.d" on page 315

# NLMNLCHFw.d Format

Writes the monetary format of the local expression for Liechtenstein and Switzerland.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLCHF***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=fr_CH;
x=put(-1234.56789,nlmnlchf32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| fr_CH | SFr.-1 234.57 |
| en_US | (SFr.1,234.57) |
| es_ES | -1.234,57 SFr. |

# See Also

**Format:**

- “NLMNICHFw.d” on page 316

# NLMNLCNYw.d Format

Writes the monetary format of the local expression for China.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNLCNY***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_CN;
x=put(-1234.56789,nlmnlcny32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| zh_CN | -RMB1,234.57 |
| en_US | (RMB1,234.57) |
| es_ES | -1.234,57 RMB |

# See Also

**Format:**

- “NLMNICNYw.d” on page 318

# NLMNLCZKw.d Format

Writes the monetary format of the local expression for the Czech Republic.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLCZK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=cs_CZ;
x=put(-1234.56789,nlmnlczk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| cs_CZ | -1 234,57 Kč |
| en_US | (Kč1,234.57) |
| es_ES | -1.234,57 Kč |

## See Also

**Format:**

- "NLMNICZKw.d" on page 319

# NLMNLDKKw.d Format

Writes the monetary format of the local expression for Denmark, Faroe Island, and Greenland.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLDKK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
options locale=da_dk;
x=put(-1234.56789,nlmnldkk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| da_DK | -1.234,57 kr |
| en_US | (kr1,234.57) |
| es_ES | -1.234,57 kr |

## See Also

**Format:**

- "NLMNIDKKw.d" on page 320

# NLMNLEEKw.d Format

Writes the monetary format of the local expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

Note: Estonia uses the Euro for curency. NLMNIEEK is documented for historical purposes.

## Syntax

**NLMNLEEK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=et_EE;
x=put(-1234.56789,nlmnleek32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| et_EE | -1 234,57 kr |
| en_US | (kr1,234.57) |
| es_ES | -1.234,57 kr |

## See Also

**Format:**

- “NLMNIEEKw.d” on page 321

# NLMNLEGPw.d Format

Writes the monetary format of the local expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNLEGP***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 3

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ar_EG;
x=put(-1234.56789,nlmnlegp32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ar_EG | 1,234.57 ج.م.- |
| en_US | (1,234.57ج.م.) |
| es_ES | -1.234,57 ج.م. |

# See Also

**Format:**

- “NLMNIEGPw.d” on page 322

# NLMNLEURw.d Format

Writes the monetary format of the local expression for Austria, Belgium, Cyprus, Estonia, Finland, France, Germany, Greece, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Portugal, Slovakia, Slovenia, and Spain.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLEUR***w.d*

### Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=de_DE;
y=put(-1234.56789,nlmnleur32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| de_DE | -1.234,57 € |
| en_US | (€1,234.57) |
| es_ES | -1.234,57 € |

## See Also

**Format:**

- "NLMNIEURw.d" on page 324

# NLMNLGBPw.d Format

Writes the monetary format of the local expression for the United Kingdom.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLGBP***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=e_GB;
x=put(-1234.56789,nlmnlgbp32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| en_GB | -£1,234.57 |
| en_US | (£1,234.57) |
| es_ES | -1.234,57 £ |

## See Also

**Format:**

- "NLMNIGBPw.d" on page 325

# NLMNLHKDw.d Format

Writes the monetary format of the local expression for Hong Kong.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLHKD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_HK;
x=put(-1234.56789,nlmnlhkd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| en_HK | (HK$1,234.57) |
| en_US | (HK$1,234.57) |
| es_ES | -1.234,57 HK$ |

## See Also

**Format:**

- "NLMNIHKDw.d" on page 326

# NLMNLHRKw.d Format

Writes the monetary format of the local expression for Croatia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLHRK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hr_HR;
x=put(-1234.56789,nlmnlhrk32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| hr_HR | -1.234,57 Kn |
| en_US | (Kn1,234.57) |
| es_ES | -1.234,57 Kn |

# See Also

**Format:**

- “NLMNIHRKw.d” on page 327

# NLMNLHUFw.d Format

Writes the monetary format of the local expression for Hungary.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNLHUF***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hu_HU;
x=put(-1234.56789,nlmnlhuf32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| hu_HU | -1 234,57 Ft |
| en_US | (Ft1,234.57) |
| es_ES | -1.234,57 Ft |

# See Also

**Format:**

- “NLMNIHUFw.d” on page 328

# NLMNLIDRw.d Format

Writes the monetary format of the local expression for Indonesia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLIDR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=id_ID;
x=put(-1234.56789,nlmnlidr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| id_ID | -Rp1.234,57 |
| en_US | (Rp1,234.57) |
| es_ES | -1.234,57 Rp |

## See Also

**Format:**

- "NLMNIIDRw.d" on page 330

# NLMNLILSw.d Format

Writes the monetary format of the local expression for Israel.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLILS***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=he_IL;
x=put(-1234.56789,nlmnlils32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| he_IL | -1,234.57 ח"ש |
| en_US | (1,234.57ח"ש) |
| es_ES | -1.234,57 ח"ש |

## See Also

**Format:**

- "NLMNIILSw.d" on page 331

# NLMNLINRw.d Format

Writes the monetary format of the local expression for India.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLINR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=hi_IN;
x=put(-1234.56789,nlmnlinr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| hi_IN | -Rs 1,234.57 |
| en_US | (Rs1,234.57) |
| es_ES | -1.234,57 Rs |

## See Also

**Format:**

- "NLMNIINRw.d" on page 332

# NLMNLJPYw.d Format

Writes the monetary format of the international expression for Japan.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLJPY***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 0

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ja_JP;
x=put(-1234.56789,nlmnljpy32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ja_JP | -JPY1,234.57 |
| en_US | (JPY1,234.57) |
| es_ES | -1.234,57 JPY |

# See Also

**Format:**

- “NLMNIJPYw.d” on page 333

# NLMNLKRWw.d Format

Writes the monetary format of the local expression for South Korea.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNLKRW***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 0

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ko_KR;
x=put(-1234.56789,nlmnlkrw32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ko_KR | -₩1,234.57 |
| en_US | (₩1,234.57) |
| es_ES | -1.234,57 ₩ |

# See Also

**Format:**

- "NLMNIKRWw.d" on page 334

# NLMNLLTLw.d Format

Writes the monetary format of the local expression for Lithuania.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLLTL***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=lt_LT;
x=put(-1234.56789,nlmnlltl32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| lt_LT | -1 234,57 Lt |
| en_US | (Lt1,234.57) |
| es_ES | -1.234,57 Lt |

## See Also

**Format:**

- "NLMNILTLw.d" on page 335

# NLMNLLVLw.d Format

Writes the monetary format of the local expression for Latvia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLLVL***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=;
x=put(-1234.56789,nlmnllvl32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| lv_LV | -1 234,57 Ls |
| en_US | (Ls1,234.57) |
| es_ES | -1.234,57 Ls |

## See Also

**Format:**

- "NLMNILVLw.d" on page 337

# NLMNLMOPw.d Format

Writes the monetary format of the local expression for Macau.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLMOP***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_MO;
x=put(-1234.56789,nlmnlmop32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| zh_MO | -P1,234.57 |
| en_US | (P1,234.57) |
| es_ES | -1.234,57 P |

## See Also

**Format:**

- "NLMNIMOPw.d" on page 338

# NLMNLMXNw.d Format

Writes the monetary format of the local expression for Mexico.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLMXN***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=es_MX;
x=put(-1234.56789,nlmnlmxn32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| es_MX | -1,234.57 MX$ |
| en_US | (MX$1,234.57) |
| es_ES | -1.234,57 MX$ |

# See Also

**Format:**

- “NLMNIMXNw.d” on page 339

# NLMNLMYRw.d Format

Writes the monetary format of the local expression for Malaysia.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNLMYR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ms_MY;
x=put(-1234.56789,nlmnlmyr32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ms_MY | (R1,234.57) |
| en_US | (R1,234.57) |
| es_ES | -1.234,57 R |

# See Also

**Format:**

- “NLMNIMYRw.d” on page 340

# NLMNLNOKw.d Format

Writes the monetary format of the local expression for Norway.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLNOK***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=no_NO;
x=put(-1234.56789,nlmnlnok32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| no_NO | -kr 1 234,57 |
| en_US | (kr1,234.57) |
| es_ES | -1.234,57 kr |

## See Also

**Format:**

- "NLMNINOKw.d" on page 341

# NLMNLNZDw.d Format

Writes the monetary format of the local expression for New Zealand.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLNZD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=en_NZ;
x=put(-1234.56789,nlmnlnzd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| en_NZ | (NZ$1,234.57) |
| en_US | (NZ$1,234.57) |
| es_ES | -1.234,57 NZ$ |

## See Also

**Format:**

- "NLMNINZDw.d" on page 343

# NLMNLPLNw.d Format

Writes the monetary format of the local expression for Poland.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLPLN***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=pl_PL;
x=put(-1234.56789,nlmnlpln32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| pl_PL | -1 234,57 zł |
| en_US | (zł1,234.57) |
| es_ES | -1.234,57 zł |

## See Also

**Format:**

- "NLMNIPLNw.d" on page 344

# NLMNLRUBw.d Format

Writes the monetary format of the local expression for Russia.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLRUB***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=ru_RU;
x=put(-1234.56789,nlmnlrub32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| ru_RU | -1 234,57 руб |
| en_US | (руб1,234.57) |
| es_ES | -1.234,57 руб |

# See Also

**Format:**

- “NLMNIRUBw.d” on page 345

# NLMNLSEKw.d Format

Writes the monetary format of the local expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNLSEK***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=sv_SV;
x=put(-1234.56789,nlmnlsek32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| sv_SE | -1 234,57 kr |
| en_US | (kr1,234.57) |
| es_ES | -1.234,57 kr |

# See Also

**Format:**

- “NLMNISEKw.d” on page 346

# NLMNLSGDw.d Format

Writes the monetary format of the local expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLSGD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_SG;
x=put(-1234.56789,nlmnlsgd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| zh_SG | -SG$1,234.57 |
| en_US | (SG$1,234.57) |
| es_ES | -1.234,57 SG$ |

## See Also

**Format:**

- "NLMNISGDw.d" on page 347

# NLMNLTHBw.d Format

Writes the monetary format of the local expression for Thailand.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLTHB***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 12

**Range** 8–32

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=th_TH;
x=put(-1234.56789,nlmnlthb32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| th_TH | ฿-1,234.57 |
| en_US | (฿1,234.57) |
| es_ES | -1.234,57 ฿ |

## See Also

**Format:**

- "NLMNITHBw.d" on page 349

# NLMNLTRYw.d Format

Writes the monetary format of the local expression for Turkey.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLTRY***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 4

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=tr_TR;
x=put(-1234.56789,nlmnltry32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| tr_TR | -1.234,57 YTL |
| en_US | (YTL1,234.57) |
| es_ES | -1.234,57 YTL |

## See Also

**Format:**

- "NLMNITRYw.d" on page 350

# NLMNLTWDw.d Format

Writes the monetary format of the local expression for Taiwan.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLTWD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

## Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=zh_TW;
x=put(-1234.56789,nlmnltwd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| zh_TW | -NT$1,234.57 |
| en_US | (NT$1,234.57) |
| es_ES | -1.234,57 NT$ |

## See Also

**Format:**

- "NLMNITWDw.d" on page 351

# NLMNLUSDw.d Format

Writes the monetary format of the local expression for Puerto Rico and the United States.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNLUSD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=es_US;
x=put(-1234.56789,nlmnlusd32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| es_US | -1,234.57 US$ |
| en_US | (US$1,234.57) |
| es_ES | -1.234,57 US$ |

# See Also

**Format:**

- “NLMNIUSDw.d” on page 352

# NLMNLZARw.d Format

Writes the monetary format of the local expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNLZAR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 12

**Range** 8–32

***d***

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

**Default** 2

**Range** 0–28

# Example

This example is executed on a UTF-8 SAS session.

The output value depends on the locale. You can set the locale option using this code.

```
Options locale=af_ZA;
x=put(-1234.56789,nlmnlzar32.2);
```

This table shows the output for the local value. Three locales are specified: the currency/locale of the format, en_US, and es_ES.

| Locale | Formatted value |
|---|---|
| af_ZA | -R1 234,57 |
| en_US | (R1,234.57) |
| es_ES | -1.234,57 R |

# See Also

**Format:**

- “NLMNIZARw.d” on page 353

# NLMNYw.d Format

Writes the monetary format of the local expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLMNY***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 9

**Range** 1–32

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

**Default** 0

**Range** 0–31

## Details

The NLMNY*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLMNY*w.d* format writes numeric values by using the currency symbol, the thousands separator, and the decimal separator that is used by the locale.

**Note:** The NLMNY*w.d* format does not convert currency format. Therefore, the value of the formatted number should equal the currency of the current locale value.

## Comparisons

The NLMNY*w.d* and NLMNYI*w.d* formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNYI*w.d* format uses three-letter international currency codes, such as USD, while NLMNY*w.d* format uses local currency symbols, such as $.

The NLMNY*w.d* format is similar to the DOLLAR*w.d* format, except that the NLMNY*w.d* format is locale-specific.

## Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmny32.2);
y=put(-1234.56789,dollar32.2);
```

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `put x=;` | `($1,234.57)` |
| `put y=;` | `$-1,234.57` |

## See Also

**Format:**

- "NLMNYIw.d" on page 400

**Informats:**

- "NLMNYw.d" in *SAS National Language Support (NLS): Reference Guide*
- "NLMNYIw.d" in *SAS National Language Support (NLS): Reference Guide*

# NLMNYIw.d Format

Writes the monetary format of the international expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLMNYI***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 9

**Range** 1–32

***d***

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

**Default** 0

**Range** 0–31

# Details

The NLMNYI*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLMNYI*w.d* format writes numeric values by using the international currency code, and locale-dependent thousands and decimal separators. The position of international currency code is also locale dependent.

**Note:** The NLMNYI*w.d* format does not convert currency format. Therefore, the value of the formatted number should equal the currency of the current locale value.

# Comparisons

The NLMNY*w.d* and NLMNYI*w.d* formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNYI*w.d* format uses three-letter international currency codes, such as USD, while NLMNY*w.d* format uses local currency symbols, such as $.

# Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnyi32.2);
y=put(-1234.56789,nlmny32.2);
```

```
z=put(-1234.56789,dollar32.2);
```

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `put x=;` | `(USD1,234.57)` |
| `put y=;` | `($1,234.57)` |
| `put z=;` | `$-1,234.57` |

## See Also

**Format:**

- "NLMNYw.d" on page 399

**Informats:**

- "NLMNYw.d" in *SAS National Language Support (NLS): Reference Guide*
- "NLMNYIw.d" in *SAS National Language Support (NLS): Reference Guide*

# NLNUMw.d Format

Writes the numeric format of the local expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLNUM***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 6

**Range** 1–32

***d***

specifies to divide the number by $10^d$. If the data contains decimal separators, the *d* value is ignored.

**Default** 0

**Range** 0–31

# Details

The NLMUM*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLNUM*w.d* format writes numeric values by using the thousands separator and the decimal separator that is used by the locale.

# Comparisons

The NLNUM*w.d* format writes the numeric value with locale-dependent thousand and decimal separators. The NLNUMI*w.d* format writes the numeric value with a comma (,) as thousands separator and a period (.) as a decimal separator.

If the *w* or *d* values are not large enough to generate a formatted number, the NLNUM*w.d* format uses an algorithm that prints the thousands-separator characters whenever possible, even if some decimal precision is lost.

# Example

```
x=put(-1234356.7891,nlnum32.2);
```

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `options LOCALE=English_UnitedStates;`<br>`put x=;` | `-1,234,356.79` |
| `options LOCALE=German_Germany;`<br>`put x=;` | `-1.234.356,79` |

# See Also

**Format:**

- “NLNUMIw.d” on page 404

**Informats:**

- "NLNUMw.d" in *SAS National Language Support (NLS): Reference Guide*
- "NLNUMIw.d" in *SAS National Language Support (NLS): Reference Guide*

# NLNUMIw.d Format

Writes the numeric format of the international expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLNUMI***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 6

**Range** 1–32

***d***

specifies to divide the number by $10^d$. If the data contains decimal points, the *d* value is ignored.

**Default** 0

**Range** 0–31

## Details

The NLNUMI*w.d* informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLNUMI*w.d* format writes numeric values by using a comma (,) as thousands separator and a period (.) as a decimal separator for all locales.

## Comparisons

The NLNUMI*w.d* format writes the numeric data of the international expression in the specified locale. The NLNUMI*w.d* format writes the numeric value with a comma (,) as thousands separator and a period (.) as a decimal separator.

If the *w* or *d* values are not large enough to generate a formatted number, the NLNUM*w.d* format uses an algorithm that prints the thousands-separator characters whenever possible, even if some decimal precision is lost.

## Example

```
x=put(-1234356.7891,nlnumi32.2);
```

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `options LOCALE=English_UnitedStates;`<br>`put x=;` | `-1,234,356.79` |
| `options LOCALE=German_Germany;`<br>`put x=;` | `-1,234,356.79` |

## See Also

**Format:**

- "NLNUMw.d" on page 402

**Informats:**

- "NLNUMw.d" in *SAS National Language Support (NLS): Reference Guide*
- "NLNUMIw.d" in *SAS National Language Support (NLS): Reference Guide*

# NLPCTw.d Format

Writes percentage data of the local expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

# Syntax

**NLPCT***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 6

**Range** 4–32

***d***

specifies to divide the number by $10^d$. If the data contains decimal separators, the *d* value is ignored.

**Default** 0

**Range** 0–31

# Comparisons

The NLPCT*w.d* format writes percentage data of the local expression in the specified locale. The NLPCT*w.d* format writes the percentage value with locale-dependent thousand and decimal separators. The NLPCTI*w.d* format writes the percentage value with a comma (,) as thousands separator and a period (.) as a decimal separator.

The NLPCT*w.d* format is similar to the PERCENT*w.d* format except the NLPCT*w.d* format is locale-specific.

# Example

```
x=put(-12.3456789,nlpct32.2);
y=put(-12.3456789,nlpcti32.2);
z=put(-12.3456789,percent32.2);
```

| Statements | Results |
|---|---|
| | `----+-----------1` |
| `options LOCALE=English_UnitedStates;`<br>`put x=;`<br>`put y=;`<br>`put z=;` | `-1,234.57%`<br>`-1,234.57%`<br>`(  1234.57%)` |
| `options LOCALE=German_Germany;` | `-1.234,57%` |

| Statements | Results |
|---|---|
| put x=; | -1,234.57% |
| put y=; | (1234.57%) |
| put z=; | |

## See Also

**Format:**

- "NLPCTIw.d" on page 407

**Informats:**

- "NLPCTw.d" in *SAS National Language Support (NLS): Reference Guide*
- "NLPCTIw.d" in *SAS National Language Support (NLS): Reference Guide*

# NLPCTIw.d Format

Writes percentage data of the international expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLPCTI***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 6

**Range** 4–32

***d***

specifies to divide the number by $10^d$. If the data contains decimal separators, the *d* value is ignored.

**Default** 0

**Range** 0–31

## Comparisons

The NLPCTI*w.d* format writes percentage data of the international expression in the specified locale. The NLPCT*w.d* format writes the percentage value with locale-dependent thousand and decimal separators. The NLPCTI*w.d* format writes the percentage value with a comma (,) as thousands separator and a period (.) as a decimal separator.

The NLPCT*w.d* format is similar to the PERCENT*w.d* format except the NLPCT*w.d* format is locale-specific.

## Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-12.3456789,nlpcti32.2);
y=put(-12.3456789,percent32.2);
```

| Statements | Results |
|---|---|
| | `----+--------1` |
| `put x=;` | `-1,234.57%` |
| `put y=;` | `(  1234.57)` |

## See Also

**Format:**

- “NLPCTw.d” on page 405

**Informats:**

- “NLPCTw.d” in *SAS National Language Support (NLS): Reference Guide*
- “NLPCTIw.d” in *SAS National Language Support (NLS): Reference Guide*

# NLPCTNw.d Format

Writes percentages, using a minus sign for negative values.

Categories: CAS
Numeric

Alignment: Right

# Syntax

**NLPCTN***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.

**Default** 0

**Range** 0–31

**Requirement** must be less than *w*

# Details

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

# Example

```
x=-0.02;
```

| Statements | Results |
|---|---|
| put x nlpctn6.; | x=-2% |
| put x percentn6.; | x=-2% |

# NLPCTPw.d Format

Writes locale-specific numeric values as percentages.

Categories: CAS
Numeric

Alignment: Right

## Syntax

**NLPCTP***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 (% ).

***d***

specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional. The thousands separator and decimal symbol for the NLPCTP format is locale-specific.

**Default** 0

**Range** 0–31

**Requirement** must be less than *w*

## Details

The NLPCTP*w.d* format multiplies values by 100, formats them, and adds a percent sign (%) to the end of the formatted value. The NLPCTP*w.d* format is similar to the The PERCENT*w.d* format, except that the thousands separator and decimal symbol for the NLPCTP*w.d* format is locale-specific.

## Example

```
x=-0.02;
```

| Statements | Results |
|---|---|
| put x nlpctp6.; | –2% |
| put x percent6.; | ( 2%) |

# NLPVALUEw.d Format

Writes p-values of the local expression that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLPVALUE***w.d*

## Syntax Description

***w***
specifies the width of the output field.

**Default** 6

**Range** 3–32

***d***
specifies to divide the number by $10^d$. If the data contains decimal separators, the *d* value is ignored.

**Default** 4

**Range** 1–30

## Example

This example uses the german_Germany locale option.

Statements:

```
options locale=german_germany;
data _null_;
   put "+--- nlpvalue min=3 default=6 max=32 ---+";
   x=0.1248;
   put x= +5 x pvalue.    +5 x nlpvalue.;
   put x= +5 x pvalue3.1  +5 x nlpvalue3.1;
   put x= +5 x pvalue20.2 +5 x nlpvalue20.2;
   put x= +5 x pvalue32.3 +5 x nlpvalue32.3;
run;
```

Results:

```
+--- nlpvalue min=3 default=6 max=32 ---+
x=0.1248   0.1248     0,1248
x=0.1248   0.1     0,1
x=0.1248                  0.12                     0,12
x=0.1248                              0.125                          0,125
```

## See Also

**Format:**

- “PVALUEw.d” on page 438

# NLSTRMONw.d Format

Writes the month name that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLSTRMON***w.d*

## Syntax Description

***w***

specifies the width of the output field

**Default** 20

**Range** 1-200

***d***

specifies the following:

- 00000001: write abbreviated form.
- 00000010: write capitalized form.

**Default** 0

**Range** 0-3

# Details

The NLSTRMONw.d format writes a SAS value, 1–12 as the name-of-the-month in the specified locale. The following examples use the English_UnitedStates locale.

- 1 = the first month (January)
- 2 = the second month (February)
- 3 = the third month (March)
- 4 = the fourth month (April)
- 5 = the fifth month (May)
- 6 = the sixth month (June)
- 7 = the seventh month (July)
- 8 = the eight month (August)
- 9 = the ninth month (September)
- 10 = the 10th month (October)
- 11 = the 11th month (November)
- 12 = the 12th month (December)

# Example

This example uses the English_UnitedStates session encoding.

| Statements | Results |
|---|---|
| Data _null_ ;<br>monnum = 1 ; /* January=1, December=12 */<br>put monnum NLSTRMON20. ;<br>put monnum NLSTRMON20.1; /* decimal .1 specified use abbreviation. */<br>put monnum NLSTRMON20.2;<br>put monnum NLSTRMON20.3; | January<br>Jan<br>JANUARY<br>JAN |

| Statements | Results |
| --- | --- |
| run; | |

# NLSTRQTRw.d Format

Writes a numeric value as the quarter-of-the-year that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLSTRQTR***w.d*

## Syntax Description

***w***

specifies the width of the output field

**Default** 20

**Range** 1–200

***d***

specifies the following:

- 00000001: write abbreviated form.
- 00000010: write capitalized form.

**Default** 0

**Range** 0–3

## Details

The NLSTRQTRw.d format writes a SAS value, 1–4 as the name-of-the-quarter for the year in the specified locale. The following examples use the English_UnitedStates locale.

- 1 = 1st quarter
- 2 = 2nd quarter
- 3 = 3rd quarter

- 4 = 4th quarter

## Example

This example uses the English_UnitedStates session encoding.

| Statements | Results |
|---|---|
| Data _null_ ;<br>qtrnum = 1 ; /* January=1, December=12 */<br>put qtrnum NLSTRQTR20. ;<br>put qtrnum NLSTRQTR20.1; /* decimal .1 specified use abbreviation. */<br>put qtrnum NLSTRQTR20.2;<br>put qtrnum NLSTRQTR20.3; run; | 1st quarter<br>Q1<br>1ST QUARTER<br>Q1 |

# NLSTRWKw.d Format

Writes a numeric value as the day-of-the-week that is appropriate for the current SAS locale.

Categories: CAS
Numeric

Alignment: Left

## Syntax

**NLSTRWK***w.d*

## Syntax Description

***w***

specifies the width of the output field

**Default** 20

**Range** 1–200

***d***

specifies the following:

- 00000001: write abbreviated form.

- 00000010: write capitalized form.

**Default** 0

**Range** 0–3

## Details

The NLSTRWKw.d format writes a SAS value, 1–7 as the name-of-the-week in the specified locale. The following examples use the English_UnitedStates locale.

- 1 = First day-of-week (Monday)
- 2 = Second day-of-week (Tuesday)
- 3 = Third day-of-week (Wednesday)
- 4 = Fourth day-of-week (Thursday)
- 5 = Fifth day-of-week (Friday)
- 6 = Sixth day-of-week (Saturday)
- 7 = Seventh day-of-week (Sunday)

## Example

This example uses the English_UnitedStates session encoding.

| Statements | Results |
|---|---|
| Data _null_ ;<br>wknum = 1 ; /* Sunday=1, Saturday=7 */<br>put wknum NLSTRWK20. ;<br>put wknum NLSTRWK20.1; /* decimal .1 specified use abbreviation. */<br>put wknum NLSTRWK20.2;<br>put wknum NLSTRWK20.3;<br>run; | Sunday<br>Sun<br>SUNDAY<br>SUN |

# NLTIMAPw. Format

Writes a SAS time value as a time value with a.m. or p.m. that is appropriate for the current SAS locale.

Categories: CAS
Date and Time

Time

Alignment: Left

# Syntax

**NLTIMAP***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 10

**Range** 4–200

# Details

The NLTIMAP format might produce inaccurate localized output when using the default width with some encoding and locale combinations because the date and time names are too long. Please refer to Exceptions for Date and Time Default Widths for information about recommended widths for locale and encoding combinations. You might need to use the recommended width.

# Comparisons

The NLTIMAP*w*. format is similar to the TIMEAMPM*w*. format, except that the NLTIMAP*w*. format is locale-specific.

NLTIMAP also writes datetime values.

# Example

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `options locale=English_UnitedStates;`<br>`put time nltimap.;` | `4:24:43 PM` |

| Statements | Results |
| --- | --- |
| `options locale=German_Germany;`<br>`put time nltimap14.;` | `4:24:43 nachm` |

## See Also

**Format:**

- "NLTIMEw." on page 418

# NLTIMEw. Format

Writes a SAS time value as a time value that is appropriate for the current SAS locale.

Categories: CAS
Date and Time
Time

Alignment: Left

## Syntax

**NLTIME***w*.

## Syntax Description

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

**Default** 20

**Range** 10–200

## Comparisons

The NLTIME*w*. format is similar to the TIME*w*. format, except that the NLTIME*w*. format is locale-specific.

NLTIME also writes datetime values.

## Example

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `options locale=English_UnitedStates;`<br>`put time nltime.;` | `4:24:43` |
| `options locale=German_Germany;`<br>`put time nltime.;` | `16.24` |

## See Also

**Format:**

- "NLTIMAPw." on page 416

# NUMXw.d Format

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

Category: Numeric

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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

```
data one;
   input x;
   datalines;
896.48
64.89
3064.10
;
run;

data two;
   set one;
   put x numx10.2;
run;
```

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

## See Also

**Formats:**

- "w.d" on page 476

**Informats:**

- "NUMXw.d" on page 654

# OCTALw. Format

Converts numeric values to octal representation.

Categories: CAS
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

```
data one;
   x=3592;
   put x octal6.;
run;
```

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

# ODDSRw.d Format

Writes odds ratios.

Category: Numeric

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

## Syntax

**OODSR***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 8

**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** 0–30

**Requirement** *d* must be less than *w*.

# Comparisons

The ODDSR*w.d* format follows the rules for the *w.d* format, except in these conditions:

- values from –1e-12 to $10^{-d}$ –eps are displayed as less than “0.0...01”
- values that are greater than or equal to $10^{w-d-2}$ are displayed as greater than “999.999”

# Example

```
data one;
   input x;
   datalines;
   0.00001
   0.0001
   0.001
   0.01
   0.1
   1
   10
   100
   1000
   I

   _
   M
   .
   -1
;
run;

data two;
   set one;
   put x oddsr8.3;
run;
```

| Value of x | Result |
|---|---|
| | ----+----1 |
| 0.00001 | <0.001 |
| 0.0001 | <0.001 |
| 0.001 | 0.001 |
| 0.01 | 0.010 |
| 0.1 | 0.100 |

| Value of x | Result |
|---|---|
| 1 | 1.000 |
| 10 | 10.000 |
| 100 | 100.000 |
| 1000 | >999.999 |
| I | >999.999 |
| – | |
| M | |
| . | . |
| -1 | ******** |

# PDw.d Format

Writes data in packed decimal format.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

See: "PD Format: UNIX" in *SAS Companion for UNIX Environments*
"PDw.d: 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

```
data a;
   x=128;
   y=put(x,pd4.);
   put y $hex8.;
run;
```

| Value of x | Result [1] |
|---|---|
| | `----+----1` |
| 128 | `00000128` |

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

# PDJULGw. Format

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

Category: Date

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 2-byte representation of the four-digit Gregorian year.

*ddd*
is the 1 1/2-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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

## Example

```
date = '15sep2018'd;
   juldate = put(date,pdjulg4.);
   put juldate $hex8.;
run;
```

SAS writes the following results to the log:

```
20182587F
```

## See Also

**Formats:**

- "JULDAYw." on page 230
- "JULIANw." on page 232
- "PDJULIw." on page 427

**Functions:**

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

**Informats:**

- "JULIANw." on page 645
- "PDJULGw." on page 658
- "PDJULIw." on page 660

**System Options:**

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

# PDJULIw. Format

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

Category: Date

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 1-byte representation of a two-digit integer that represents the century.

*yy*
is the 1-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 four-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 1 1/2-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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

# Examples

# Example 1

```
data _null_;
   date = '15jun2018'd;
   juldate = put(date,pdjuli4.);
   put juldate $hex8.;
run;
```

SAS writes the following results to the log:

```
0118258F
```

## Example 2

```
data _null_;
   date = '31dec2018'd;
   juldate = put(date,pdjuli4.);
   put juldate $hex8.;
run;
```

SAS writes the following results to the log:

```
0118365F
```

## See Also

**Formats:**

- “JULDAYw.” on page 230
- “JULIANw.” on page 232
- “PDJULGw.” on page 426

**Functions:**

- “DATEJUL” in *SAS Functions and CALL Routines: Reference*
- “JULDATE” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “JULIANw.” on page 645
- “PDJULGw.” on page 658
- “PDJULIw.” on page 660

**System Options:**

- “YEARCUTOFF=” in *SAS System Options: Reference*

# PERCENTw.d Format

Writes numeric values as percentages.

| | |
|---|---|
| Categories: | CAS<br>Numeric |
| Alignment: | Right |
| Interaction: | When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV=” in *SAS System Options: Reference*. |

# 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 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. Negative values are enclosed in parentheses.

# Example

```
data one;
   input gain;
   datalines;
.1
1.2
-0.05
;
run;

data two;
   set one;
   put @10 gain percent10.;
run;
```

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

## See Also

**Formats:**

- "PERCENTNw.d" on page 431

# PERCENTNw.d Format

Produces percentages, using a minus sign for negative values.

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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

```
data one;
   input gain;
   datalines;
-0.1
0.2
.8
-0.05
-6.3
;
run;

data two;
   set one;
   put @10 gain percentn10.;
run;
```

| Value of gain | Result |
|---|---|
| –0.1 | -10% |
| .2 | 20% |
| .8 | 80% |

| Value of gain | Result |
|---|---|
| –0.05 | `-5%` |
| –6.3 | `-630%` |

## See Also

**Format:**

- “PERCENTw.d” on page 429

# PIBw.d Format

Writes positive integer binary (fixed-point) values.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

See: “PIB Format: UNIX” in *SAS Companion for UNIX Environments*
“PIBw.d: 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** 0–10

# 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 more information about byte ordering, see "Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms" on page 37 .

# 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 contains 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 "Writing Data Generated on Big Endian and Little Endian Platforms" on page 37.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

# Example

```
data _null_;
    x=12;
    y=put(x,pib1.);
    put y $hex2.;
 run;
```

| Value of x | Result [1] |
|---|---|
| | `----+----1` |

| Value of x | Result [1] |
|---|---|
| 12 | 0C |

1 The result is a hexadecimal representation of a 1-byte binary number written in positive integer binary format, which occupies one column of the output field.

## See Also

**Formats:**

- "PIBRw.d" on page 435

# PIBRw.d Format

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

Category: Numeric

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 in 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 37 .

# 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 contains values between hexadecimal 80 and hexadecimal FF, where the high-order bit can be misinterpreted as a negative sign.
- In 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 "Writing Data Generated on Big Endian and Little Endian Platforms" on page 37.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

# Example

```
data _null_;
   x=128;
   y=put(x,pibr2.);
   put y $hex4.;
run;
```

| Value of x | Result [1] |
|---|---|
| | `----+----1` |
| 128 | `8000` |

1 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" on page 664

# PKw.d Format

Writes data in unsigned packed decimal format.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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

```
data _null_;
   x=128;
   y=put(x,pk4.);
   put y $hex8.;
run;
```

| Value of x | Result [1] |
|---|---|
| | `----+----1` |
| 128 | `00000128` |

1 The result is a hexadecimal representation of a 4-byte number written in packed decimal format. Each byte occupies one column of the output field.

# PVALUEw.d Format

Writes *p*-values.

Category: Numeric

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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

**Requirement** *d* must be less than *w*.

# Comparisons

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

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

# Example

```
data one;
   input x;
   datalines;
.05
.000001
0
.0123456
;
run;

data two;
   set one;
   put x pvalue6.4;
run;
```

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

# QTRw. Format

Writes date values as the quarter of the year.

Categories: CAS
Date

Alignment: Right

## Syntax

**QTR***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 1

**Range** 1–32

## Example

```
data one;
   x='05mar2018'd;
   put x qtr.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put x qtr.;` | `1` |

## See Also

**Formats:**

- "QTRRw." on page 441

# QTRRw. Format

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

Categories: CAS
Date

Alignment: Right

## Syntax

**QTRR***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 3

**Range** 3–32

## Example

```
data one;
   x='05sep2018'd;
   put x qtrr.;
run;
```

The example table uses the input value of 21432, which is the SAS date value that corresponds to September 5, 2018.

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

## See Also

**Formats:**

- "QTRw." on page 440

# RBw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

See: "RB Format: UNIX" in *SAS Companion for UNIX Environments*
"RBw.d: 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 the data. 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 8-byte (double-precision) number rather than a true 4-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

```
data one;
   x=128;
   y=put(x,rb8.);
   put y $hex16.;
run;
```

| Value of x | Result [1] |
|---|---|
| | `----+---1----+----2` |

| Value of x | Result [1] |
|---|---|
| 128 | 0000000000006040 |

1 The result is a hexadecimal representation of an 8-byte real binary number as it looks on an IBM mainframe. Each byte occupies one column of the output field.

# ROMANw. Format

Writes numeric values as Roman numerals.

Categories: CAS
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

```
data one;
   input x;
datalines;
   2018
   52.3
;
run;
```

```
data two;
   set one;
   put x roman10.;
run;
```

| Value of x | Result |
|---|---|
| 2018 | MMXVIII |
| 52.3 | LII |

# S370FFw.d Format

Writes native standard numeric data in IBM mainframe format.

Category: Numeric

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 1 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

```
data _null_;
   x=12345;
   y=put(x,s370ff5.);
   put y $hex10.;
run;
```

| Value of x | Result [1] |
|---|---|
| | `----+----1` |
| 12345 | `F1F2F3F4F5` |

1 The result is the hexadecimal representation for the integer.

## See Also

**Formats:**

- "$EBCDICw." on page 109
- "w.d" on page 476

# S370FIBw.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 is 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 37 .

# 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 "Writing Data Generated on Big Endian and Little Endian Platforms" on page 37.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

## Example

```
data _null_;
   x=128;
   y=put(x,s370fib4.);
   put y $hex8.;
run;
```

| Value of x | Result [1] |
|---|---|
| | ----+----1 |
| 128 | 00000080 |

1 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" on page 448
- "S370FPIBw.d" on page 453

# S370FIBUw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 is 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 37 .

# 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, whereas 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 "Writing Data Generated on Big Endian and Little Endian Platforms" on page 37.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

# Example

```
data one;
```

```
   input x;
   datalines;
245
-245
;
run;

data two;
   set one;
   y=put(x,s370fibu2.);
   put y $hex4.;
run;
```

| Value of x | Result [1] |
|---|---|
| 245 | F5 |
| -245 | F5 |

1 The result is a hexadecimal representation of a 1-byte integer binary number. Each byte occupies one column of the output field.

## See Also

### Formats

- “S370FIBw.d” on page 446
- “S370FPIBw.d” on page 453

# S370FPDw.d Format

Writes packed decimal data in IBM mainframe format.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV=” in *SAS System Options: Reference*.

## 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. |
| PL/I | FIXED DEC(7,0) |
| COBOL | COMP-3 PIC S9(7) |
| IBM 370 assembler | PL4 |

# Example

```
data _null_;
   x=128;
   y=put(x,s370fpd4.);
   put y $hex8.;
run;
```

| Value of x | Result [1] |
|---|---|
| | `----+----1` |
| 128 | `0000128C` |

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

# S370FPDUw.d Format

Writes unsigned packed decimal data in IBM mainframe format.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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 *n* is the number of digits.

## Example

```
data one;
   input x;
   datalines;
123
-123
;
run;
   data two;
   set one;
   y=put(x,s370fpdu4.);
   put y $hex8.;
run;
```

| Value of x | Result [1] |
|---|---|
| 123 | `123F` |
| -123 | `123F` |

1 The result is a hexadecimal representation of a binary number written in packed decimal format. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the output field.

# S370FPIBw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 is 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 37 .

# 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 "Writing Data Generated on Big Endian and Little Endian Platforms" on page 37.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

## Example

```
data _null_;
   x=12;
   y=put(x,s370fpib1.);
   put y $hex2.;
run;
```

| Value of x | Result [1] |
|---|---|
| | `----+----1` |
| 12 | `0C` |

1 The result is a hexadecimal representation of a 1-byte binary number written in positive integer binary format, which occupies one column of the output field.

## See Also

**Formats:**

- "S370FIBw.d" on page 446
- "S370FIBUw.d" on page 448

# S370FRBw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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

```
data _null_;
   input x;
   y=put(x,s370frb6.);
   put y $hex8.;
   datalines;
128
-128
;
run;
```

| Value of x | Result [1] |
|---|---|
| 128 | 42800000 |
| -123 | C2800000 |

1 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 1 byte of binary data, and each byte corresponds to one column of the output field.

# S370FZDw.d Format

Writes zoned decimal data in IBM mainframe format.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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

```
data one;
   input x;
datalines;
123
-123
;
run;
data two;
   set one;
   y=put(x,s370fzd3.);
   put y $hex6.;
run;
```

| Value of x | Result [1] |
|---|---|
| 123 | F1F2C3 |
| -123 | F1F2D3 |

1 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 1 byte of binary data, and each byte corresponds to one column of the output field.

# S370FZDLw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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 *n* is the number of digits.

# Example

```
data one;
   input x;
   datalines;
123
```

```
-123
;
run;

data two;
   set one;
   y=put(x,s370fzdl3.);
   put y $hex6.;
run;
```

| Value of x | Result [1] |
|---|---|
| 123 | C1F2F3 |
| -123 | D1F2F3 |

1 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 1 byte of binary data, and each byte corresponds to one column of the output field.

# S370FZDSw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 *n* is the number of digits.

## Example

```
data one;
   input x;
   datalines;
123
-123
;
run;
   data two;
   set one;
   y=put(x,s370fzds3.);
   put y $hex6.;
run;
```

| Value of x | Result [1] |
|---|---|
| 123 | 4EF1F2F3 |
| -123 | 60F1F2F3 |

1 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 1 byte of binary data, and each byte corresponds to one column of the output field.

# S370FZDTw.d Format

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

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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 *n* is the number of digits.

# Example

```
data one;
   input x;
   datalines;
123
```

```
-123
;
run;
data two;
   set one;
   y=put(x,s370fzdt4.);
   put y $hex8.;
run;
```

| Value of x | Result [1] |
|---|---|
| 123 | F1F2F34E |
| -123 | F1F2F360 |

1 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 1 byte of binary data, and each byte corresponds to one column of the output field.

# S370FZDUw.d Format

Writes unsigned zoned decimal data in IBM mainframe format.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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 *n* is the number of digits.

## Example

```
data one;
   input x;
   datalines;
123
-123
;
run;
data two;
   set one;
   y=put(x,s370fzdu3.);
   put y $hex6.;
run;
```

| Value of x | Result [1] |
|---|---|
| 123 | F1F2F3 |
| -123 | F1F2F3 |

1 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 1 byte of binary data, and each byte corresponds to one column of the output field.

# SSNw. Format

Writes Social Security numbers.

Category: Numeric

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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

```
data one;
id=263878439;
put id ssn.;
run;
```

| Value of id | Result |
|---|---|
| 263878439 | 263-87-8439 |

# TIMEw.d Format

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

Categories: CAS
Time

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

Note: The value of TIME can be a SAS time value, but it can also represent a time duration. Therefore, the value of TIME can exceed 23:59:59, and it can also be a negative value.

# 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, 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 0 for a single-hour digit.

# Examples

## Example 1

```
data one;
   mytime='16:24:43't;
   put mytime time.;
run;
```

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 mytime time.;` | `16:24:43` |

## Example 2

```
data one;
   mytime='8:00:00't *3;
   put mytime time.;
run;
```

This example uses the input value of '8:00:00't and a multiplier of 3 for the total (`*3;`), which indicates that the total time duration (such as 3 days of work) is 24 hours.

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mytime time.;` | `24:00:00` |

## Example 3

```
data one;
   x='4:00:00't;
   y='3:00:00't;
   diff=y-x;
   put diff time.;
run;
```

This example uses the input values of x='4:00:00't and y='3:00:00't and determines that the value of y is one hour less than the value of x (`diff = y-x;`).

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put diff time8.;` | `-1:00:00` |

## See Also

**Formats:**

- "HHMMw.d" on page 221
- "HOURw.d" on page 224
- "MMSSw.d" on page 240
- "TODw.d" on page 470

**Functions:**

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

**Informats:**

- "TIMEw." on page 702

# TIMEAMPMw.d Format

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

Categories: CAS
Date and Time
Time

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

# 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. The letters h, m, and s are defined as follows:

*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 TIMEAMPM*w.d* format is similar to the TIME*w.d* format, except that TIMEAMPM*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

```
data one;
   begin='16:24:43't;
   put begin timeampm3.;
   put begin timeampm5.;
   put begin timeampm8.;
   put begin timeampm11.;
run;
```

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----+` |
| `put begin timeampm3.;` | `PM` |
| `put begin timeampm5.;` | `4 PM` |
| `put begin timeampm8.;` | `4:24 PM` |
| `put begin timeampm11.;` | `4:24:43 PM` |

## See Also

**Formats:**

- “TIMEw.d” on page 465

# TODw.d Format

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

Categories: CAS
Date and Time

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

Note: You can specify a value greater than 24:00:00 for the TOD format, but it takes the MOD of the value and 24:00:00. For example, if you give TOD a value of '25:00:00't, it formats the value as 1:00:00.

# Syntax

**TOD***w.d*

## Syntax Description

***w***

specifies the width of the output field.

**Default** 8

**Range** 2–20

**Tip** SAS writes a 0 for a zero hour if the specified width is sufficient. For example, the times 02:30 or 00:30 have a zero in the hour digits.

***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** *d* 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 0 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

```
data one;
   mydt1='24aug2018 2:20:23 pm'dt;
   mydt2='1:30't;
   mydt3='8:54:43 am't;
   put mydt1 tod5.;
   put mydt2 tod9.;
   put mydt3 tod9.;
run;
```

In this example, the SAS datetime value 1850739623 corresponds to August 24, 2018 at 2:20:23 p.m.

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydt1 tod5.;` | `14:20` |
| `put mydt2 tod9.;` | `01:30:00` |
| `put mydt3 tod9.;` | `08:54:43` |

# Example 2

In this example, the SAS time value 32083 corresponds to 8:54:43 a.m.

```
data _null_;
   begin = 32083;
   put begin tod9.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `begin = 32083;` | `08:54:43` |

| SAS Statement | Result |
| --- | --- |
| `put begin tod9.;` | |

## See Also

**Formats:**

- “HHMMw.d” on page 221
- “TIMEw.d” on page 465
- “TIMEAMPMw.d” on page 468

**Functions:**

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

**Informats:**

- “TIMEw.” on page 702

# VAXRBw.d Format

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

Categories: CAS
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, the VAXRB*w.d* and RB*w.d* formats are identical.

## Example

```
data one;
   x=1;
   y=put(x,vaxrb8.);
   put y $hex16.;
run;
```

| Value of x | Result [1] |
|---|---|
| 1 | 8040000000000000 |

1 The result is the hexadecimal representation for the integer.

# VMSZNw.d Format

Generates VMS and MicroFocus COBOL zoned numeric data.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## Syntax

**VMSZN***w.d*

## Syntax Description

***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. This 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 |
| 9 | 9 | –9 | y |

If the value to be formatted is too large to fit in a field of the specified width, the VMSZN*w.d* format handles values this way:

- 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

```
data one;
   x=1234;
   put x vmszn4.;
   put x vmszn5.1;
   put x vmszn6.2;
   put x vmszn5.;
run;
```

SAS writes the following results to the log:

```
----+----1
1234
12340
123400
01234
```

## See Also

**Formats:**

- "ZDw.d" on page 522

**Informats:**

- "VMSZNw.d" on page 708

# w.d Format

Writes standard numeric data one digit per byte.

| | |
|---|---|
| Categories: | CAS<br>Numeric |
| Alignment: | Right |
| Alias: | F*w.d* |
| Interaction: | When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*. |

# 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

```
data one;
   x=23.45;
   put x 6.3;
run;
```

SAS writes the following results to the log:

```
23.450
```

# WEEKDATEw. 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

Alignment: Right

Restriction: Use the NLDATE*w*. format on the CAS server instead of the WEEKDATE*w* format.

## 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 21349, which is the SAS date value that corresponds to June 14, 2018.

```
data one;
   mydate='14jun2018'd;
   put mydate weekdate3.;
   put mydate weekdate9.;
   put mydate weekdate15.;
   put mydate weekdate17.;
   run;
```

| SAS Statement | Result |
|---|---|
| | ----+----1----+----2 |
| put mydate weekdate3.; | Thu |
| put mydate weekdate9.; | Thursday |
| put mydate weekdate15.; | Thu, Jun 14, 18 |
| put mydate weekdate17.; | Thu, Jun 14, 2018 |

# See Also

**Formats:**

- “DATEw.” on page 169
- “DDMMYYw.” on page 175
- “MMDDYYw.” on page 235
- “TODw.d” on page 470
- “WEEKDATXw.” on page 480
- “YYMMDDw.” on page 498

**Functions:**

- “JULDATE” in *SAS Functions and CALL Routines: Reference*
- “MDY” in *SAS Functions and CALL Routines: Reference*
- “WEEKDAY” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “DATEw.” on page 617
- “DDMMYYw.” on page 621
- “MMDDYYw.” on page 649

■ "YYMMDDw." on page 720

# WEEKDATXw. Format

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

Categories: CAS
Date

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, 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 21349, which is the SAS date value that corresponds to June 14, 2018.

```
data one;
   mydate='14jun2018'd;
   put mydate weekdatx.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+----2----+----3` |
| `put mydate weekdatx.;` | `Thursday, 14 June 2018` |

# See Also

**Formats:**

- “DATEw.” on page 169
- “DDMMYYw.” on page 175
- “DTWKDATXw.” on page 187
- “MMDDYYw.” on page 235
- “TODw.d” on page 470
- “WEEKDATEw.” on page 478
- “YYMMDDw.” on page 498

**Functions:**

- “JULDATE” in *SAS Functions and CALL Routines: Reference*
- “MDY” in *SAS Functions and CALL Routines: Reference*
- “WEEKDAY” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “DATEw.” on page 617
- “DDMMYYw.” on page 621
- “MMDDYYw.” on page 649
- “YYMMDDw.” on page 720

# WEEKDAYw. Format

Writes date values as the day of the week.

Categories: CAS
Date

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

```
data one;
   mydate='15sep2018'd;
   put mydate weekday.;
run;
```

The example table uses the input value of 21442, which is the SAS date value that corresponds to September 15, 2018.

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydate weekday.;` | `7` |

## See Also

**Formats:**

- "DOWNAMEw." on page 183

**Functions**

- "WEEKDAY" in *SAS Functions and CALL Routines: Reference*

# WEEKUw. Format

Writes a week number in decimal format by using the U algorithm.

| | |
|---|---|
| Category: | Date |
| Alignment: | Left |
| Restriction: | This format is not supported in a DATA step that runs in CAS. |

## Syntax

**WEEKU***w*.

### Syntax Description

***w***

specifies the width of the output field.

**Default** 11

**Range** 2–200

## 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 0 and maximum value of 53. For example, the fifth week of the year would be represented as 05.

Here are widths, formats, and examples:

| Width | Format | Example |
|---|---|---|
| 3-4 | Www | w01 |
| 5-6 | yyWww | 18W01 |
| 7-8 | yyWwwdd | 18W0101 |
| 9-10 | yyyyWwwdd | 2018W0101 |
| 11-200 | yyyy-Www-dd | 2018-W01-01 |

# Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53. Weeks that begin on a Monday and week 1 of the year include January 4 and the first Thursday of the year. If the first Monday of January is 2, 3, or 4, 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 (with Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading 0.

# Example

```
data one;
   sasdate='16apr2018'd;
   put sasdate weeku3.;
   put sasdate weeku5.;
   put sasdate weeku7.;
   put sasdate weeku9.;
   put sasdate weeku11.;
run;
```

| Statement | Result |
|---|---|
| | `----+----1----+` |
| `put sasdate weeku3.;` | `W15` |
| `put sasdate weeku5.;` | `18W15` |
| `put sasdate weeku7.;` | `18W1502` |
| `put sasdate weeku9.;` | `2018W1502` |
| `put sasdate weeku11.;` | `2018-W15-02` |

## See Also

**Formats:**

- "WEEKVw." on page 485
- "WEEKWw." on page 487

**Functions:**

- "WEEK" in *SAS Functions and CALL Routines: Reference*

**Informats:**

- "WEEKUw." on page 711
- "WEEKVw." on page 713
- "WEEKWw." on page 716

# WEEKVw. Format

Writes a week number in decimal format by using the V algorithm.

| | |
|---|---|
| Category: | Date |
| Alignment: | Left |
| Restriction: | This format is not supported in a DATA step that runs in CAS. |

## Syntax

**WEEKV***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 11

**Range** 2–200

# Details

The WEEKV*w*. format writes the various formats depending on the specified width. Algorithm V calculates the SAS date value. The number-of-the-week value is represented as a decimal number in the range 01–53, with a leading 0 and maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4 and the first Thursday of the year. If the first Monday of January is 2, 3, or 4, 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.

Here are the widths, formats, and examples:

| Width | Format | Example |
|---|---|---|
| 3-4 | Www | w01 |
| 5-6 | yyWww | 18W01 |
| 7-8 | yyWwwdd | 18W0101 |
| 9-10 | yyyyWwwdd | 2018W0101 |
| 11-200 | yyyy-Www-dd | 2018-W01-01 |

# Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53.Weeks that begin on a Monday and week 1 of the year include January 4 and the first Thursday of the year. If the first Monday of January is 2, 3, or 4, 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 (with Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading 0.

# Example

```
data one;
   sasdate='16apr2018'd;
   put sasdate weekv3.;
   put sasdate weekv5.;
   put sasdate weekv7.;
   put sasdate weekv9.;
   put sasdate weekv11.;
run;
```

| Statement | Result |
|---|---|
| | ----+----1----+ |
| put sasdate weekv3.; | W37 |
| put sasdate weekv5.; | 18W15 |
| put sasdate weekv7.; | 18W1502 |
| put sasdate weekv9.; | 2018W1502 |
| put sasdate weekv11.; | 2018-W15-02 |

## See Also

**Formats:**

- “WEEKUw.” on page 483
- “WEEKWw.” on page 487

**Functions:**

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

**Informats:**

- “WEEKUw.” on page 711
- “WEEKVw.” on page 713
- “WEEKWw.” on page 716

# WEEKWw. Format

Writes a week number in decimal format by using the W algorithm.

Category: Date

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## Syntax

**WEEKW***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 11

**Range** 2–200

# Details

The WEEKW*w*. format writes the various formats depending on the specified width. Algorithm W calculates the SAS date value by 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 0 and a maximum value of 53. For example, the fifth week of the year would be represented as 05.

Here are widths, formats, and examples:

| Width | Format | Example |
|---|---|---|
| 3–4 | Www | w01 |
| 5–6 | yyWww | 18W01 |
| 7–8 | yyWwwdd | 18W0101 |
| 9–10 | yyyyWwwdd | 2018W0101 |
| 11–200 | yyyy-Www-dd | 2018-W01-01 |

# Comparisons

The WEEKV*w*. format writes the week number as a decimal number in the range 01–53. Weeks that begin on a Monday and week 1 of the year include January 4 and the first Thursday of the year. If the first Monday of January is 2, 3, or 4, 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 (with Sunday as the first day of the week) as a decimal number in the range 0–53, with a leading 0.

# Example

```
data _null_;
```

```
    sasdate='16apr2018'd;
    v=put(sasdate,weekw3.);
    w=put(sasdate,weekw5.);
    x=put(sasdate,weekw7.);
    y=put(sasdate,weekw9.);
    z=put(sasdate,weekw11.);
run;
```

| Statement | Result |
|---|---|
| | ----+----1----+ |
| put sasdate weekv3.; | W16 |
| put sasdate weekv5.; | 18W16 |
| put sasdate weekv7.; | 18W1601 |
| put sasdate weekv9.; | 2018W1601 |
| put sasdate weekv11.; | 2018-W16-01 |

## See Also

**Formats:**

- “WEEKUw.” on page 483
- “WEEKVw.” on page 485

**Functions:**

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

**Informats:**

- “WEEKUw.” on page 711
- “WEEKVw.” on page 713
- “WEEKWw.” on page 716

# WORDDATEw. Format

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

Category: Date

Alignment: Right

Restriction: The WORDDATE*w*. format is not valid on the CAS server. Use the NLDATE*w.* format on the CAS server instead of the WORDDATE*w.* format.

# 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 21349, which is the SAS date value that corresponds to June 14, 2018.

```
data one;
   mydate='14jun2018'd;
   put mydate worddate3.;
   put mydate worddate9.;
   put mydate worddate12.;
   put mydate worddate20.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+----2` |
| `put mydate worddate3.;` | `Jun` |
| `put mydate worddate9.;` | `  June` |
| `put mydate worddate12.;` | `Jun 14, 2018` |
| `put mydate worddate20.;` | `  June 14, 2018` |

## See Also

**Formats:**

- "WORDDATXw." on page 491

# WORDDATXw. Format

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

Category: Date

Alignment: Right

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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. For days 1–9, the leading 0 is not displayed.

*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 21349, which is the SAS date value that corresponds to June 14, 2018.

```
data one;
   mydate='14jun2018'd;
   put mydate worddatx.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+----2` |
| `put mydate worddatx.;` | `14 June 2018` |

# See Also

**Formats:**

- “WORDDATEw.” on page 489

# WORDFw. Format

Writes numeric values as words with fractions that are shown numerically.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 is printed as eight and 20/100.

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

```
data one;
   value=2.5;
```

```
   put value wordf15.;
run;
```

| Value of value | Result |
|---|---|
| | `----+----1----+` |
| 2.5 | `two and 50/100` |

## See Also

**Formats:**

- “WORDSw.” on page 494

# WORDSw. Format

Writes numeric values as words.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

## 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 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 is printed as five and thirty hundredths. When the value's equivalent in words does not fit in the specified field, the word is truncated on the right and the last character is printed 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

```
data one;
   value=2.5;
   put value words25.;
run;
```

| Value | Result |
|---|---|
| | `----+----1----+----2----+` |
| 2.5 | `two and fifty hundredths` |

## See Also

**Formats:**

- "WORDFw." on page 493

# YEARw. Format

Writes date values as the year.

Categories: CAS
Date

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 are printed. Otherwise, the year value is printed 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

```
data one;
   mydate='14jun2018'd;
   put mydate year2.;
   put mydate year4.;
run;
```

The example table uses the input value of 21349, which is the SAS date value that corresponds to June 14, 2018.

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydate year2.;` | `18` |
| `put mydate year4.;` | `2018` |

# See Also

**Formats:**

■ "DTYEARw." on page 189

# YYMMw. 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 two or four digits.

Categories: CAS
Date

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

```
data one;
   mydate='14jun2018'd;
```

```
   put mydate yymm.;
   put mydate yymm5.;;
   put mydate yymm6.;
   put mydate yymm7.;
   put mydate yymm10.;
run;
```

The following examples use the input value of 21349, which is the SAS date value that corresponds to June 14, 2018.

| SAS Statement | Result |
|---|---|
| | ----+----1----+ |
| put mydate yymm.; | 2018M06 |
| put mydate yymm5.; | 18M06 |
| put mydate yymm6.; | 18M06 |
| put mydate yymm7.; | 2018M06 |
| put mydate yymm10.; | 2018M06 |

## See Also

**Formats:**

- “MMYYw.” on page 241
- “YYMMxw.” on page 503

# YYMMDDw. 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 two or four digits.

Categories: CAS
Date

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 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 these forms:

*yymmdd*

*<yy>yy–mm–dd*

The letters and special character are defined as follows:

*<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

```
data one;
   mydate='03apr2018'd;
   put mydate yymmdd2.;
   put mydate yymmdd3.;;
   put mydate yymmdd4.;
   put mydate yymmdd5.;
   put mydate yymmdd6.;
   put mydate yymmdd7.;
   put mydate yymmdd8.;
   put mydate yymmdd10.;
run;
```

The following examples use the input value of 21277, which is the SAS date value that corresponds to April 3, 2018.

| SAS Statement | Result |
|---|---|
| | ----+----1----+ |
| put mydate yymmdd2.; | 18 |
| put mydate yymmdd3.; | 18 |
| put mydate yymmdd4.; | 1804 |
| put mydate yymmdd5.; | 18-04 |
| put mydate yymmdd6.; | 180403 |
| put mydate yymmdd7.; | 180403 |
| put mydate yymmdd8.; | 18-04-03 |
| put mydate yymmdd10.; | 2018-04-03 |

# See Also

**Formats:**

- “DATEw.” on page 169
- “DDMMYYw.” on page 175
- “MMDDYYw.” on page 235
- “YYMMDDxw.” on page 501

**Functions:**

- “DAY” in *SAS Functions and CALL Routines: Reference*
- “MDY” in *SAS Functions and CALL Routines: Reference*
- “MONTH” in *SAS Functions and CALL Routines: Reference*
- “YEAR” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “DATEw.” on page 617
- “DDMMYYw.” on page 621
- “MMDDYYw.” on page 649

# YYMMDDxw. 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 that 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 two or four digits.

Categories: CAS
Date

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 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 these forms:

*yymmdd*

*<yy>yyxmmxdd*

The letters and special character are defined as follows:

*<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

```
data one;
   mydate='14may2018'd;
   put mydate yymmddc5.;
   put mydate yymmddd8.;;
   put mydate yymmddp10.;
   put mydate yymmddn8.;
run;
```

The following examples use the input value of 21318, which is the SAS date value that corresponds to May 14, 2018.

| SAS Statement | Result |
|---|---|
| | `----+----1----+` |
| `put mydate yymmddc5.;` | `18:05` |
| `put mydate yymmddd8.;` | `18-05-14` |
| `put mydate yymmddp10.;` | `2018.05.14` |
| `put mydate yymmddn8.;` | `20180514` |

# See Also

**Formats:**

- “DATEw.” on page 169

- "DDMMYYxw." on page 177
- "MMDDYYxw." on page 237
- "YYMMDDw." on page 498


**Functions:**

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

**Informats:**


- "YYMMDDw." on page 720


# YYMMxw. 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.

| | |
|---|---|
| Categories: | CAS<br>Date |
| Alignment: | Right |

## 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 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 these forms:

*<yy>yymm*

*<yy>yyXmm*

The letters are defined as follows:

*<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 2138, which is the SAS date value that corresponds to May 14, 2018.

```
data one;
   mydate='14may2018'd;
   put mydate yymmc5.;
   put mydate yymmd.;
   put mydate yymmn4.;
   put mydate yymmp8.;
   put mydate yymms10.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+` |
| `put mydate yymmc5.;` | `18:05` |
| `put mydate yymmd.;` | `2018-05` |
| `put mydate yymmn4.;` | `1805` |
| `put mydate yymmp8.;` | `2018.05` |
| `put mydate yymms10.;` | `2018/05` |

## See Also

**Formats:**

- “MMYYxw.” on page 243
- “YYMMw.” on page 497

# YYMONw. Format

Writes date values in the form *yymmm* or *yyyymmm*.

Categories: CAS
Date
Numeric

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

```
data one;
   mydate='14jun2018'd;
   put mydate yymon6.;
   put mydate yymon7.;
run;
```

The example table uses the input value of 21349, which is the SAS date value that corresponds to June 14, 2018.

| SAS Statement | Result |
|---|---|
| | `----+----1` |
| `put mydate yymon6.;` | `18JUN` |
| `put mydate yymon7.;` | `2018JUN` |

## See Also

**Formats:**

- "MMYYw." on page 241

# YYQw. Format

Writes date values in the form <*yy*>*yy*Q*q*, where Q is the separator, the year appears as either two or four digits, and *q* is the quarter of the year.

Categories: CAS
Date

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 21349, which is the SAS date value that corresponds to June 14, 2018.

```
data one;
   mydate='14jun2018'd;
   put mydate yyq.;
   put mydate yyq4.;
   put mydate yyq5.;
   put mydate yyq6.;
   put mydate yyq10.;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+` |
| `put mydate yyq.;` | `2018Q2` |

| SAS Statement | Result |
|---|---|
| `put mydate yyq4.;` | `18Q2` |
| `put mydate yyq5.;` | `18Q2` |
| `put mydate yyq6.;` | `2018Q2` |
| `put mydate yyq10.;` | `2018Q2` |

## See Also

**Formats:**

- "YYQxw." on page 508
- "YYQRw." on page 510

# YYQxw. 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. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits.

Categories: CAS
Date

Alignment: Right

## Syntax

**YYQ***xw*.

## Syntax Description

***x***

identifies a separator or specifies that no separator appear between the year and the quarter. These are the 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** 6

**Range** 4–32

**Interactions** When *x* is set to *N*, no separator is specified. The width range is 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 these forms:

*<yy>yyq*

*<yy>yyxq*

The letters and special character are defined as follows:

*<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 21366, which is the SAS date value that corresponds to July 01, 2018.

```
data one;
   mydate='01jul2018'd;
   put mydate yyqc4.;
   put mydate yyqd.;
```

```
   put mydate yyqn3.;
   put mydate yyqp6.;
   put mydate yyqs8.;
run;
```

| SAS Statement | Result |
|---|---|
| | ----+----1----+ |
| put mydate yyqc4.; | 18:3 |
| put mydate yyqd.; | 2018-3 |
| put mydate yyqn3.; | 183 |
| put mydate yyqp6.; | 2018.3 |
| put mydate yyqs8.; | 2018/3 |

## See Also

**Formats:**

- “YYQw.” on page 506
- “YYQRxw.” on page 512

# YYQRw. Format

Writes date values in the form <*yy*>*yy*Q*qr*, where Q is the separator, the year appears as either two or four digits, and *qr* is the quarter of the year expressed in Roman numerals.

Categories: CAS
Date

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 21366, which is the SAS date value that corresponds to July 01, 2018.

```
data one;
   mydate='01jul2018'd;
   put mydate yyqr.;
   put mydate yyqr6.;
   put mydate yyqr7.;
   put mydate yyqr8.;
   put mydate yyqr10.;
run;
```

| SAS Statement | Result |
|---|---|
| | ----+----1----+ |
| put mydate yyqr.; | 2018QIII |
| put mydate yyqr6.; | 18QIII |
| put mydate yyqr7.; | 2018QIII |
| put mydate yyqr8.; | 2018QIII |
| put mydate yyqr10.; | 2018QIII |

## See Also

**Formats:**


- "YYQw." on page 506
- "YYQRxw." on page 512


# YYQRxw. 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. The special character separates the year and the quarter or the year. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator. The year can be either two or four digits, and *qr* is the quarter of the year expressed in Roman numerals.

Categories: CAS
Date
Numeric

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 the 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 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 these forms:

*<yy>yyqr*

*<yy>yyxqr*

The letters and special character are defined as follows:

*<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 21318, which is the SAS date value that corresponds to May 14, 2018.

```
data _null_;
   input date anydtdte12.;
   put date;
   put date yyqrc6.;
   put date yyqrd.;
   put date yyqrn5.;
   put date yyqrp8.;
   put date yyqrs10.;
   datalines;
05/14/2018
;
run;
```

| SAS Statement | Result |
|---|---|
| | `----+----1----+` |
| `put date;` | `21318` |
| `put date yyqrc6.;` | `18:II` |
| `put date yyqrd.;` | `2018-II` |

| SAS Statement | Result |
|---|---|
| `put date yyqrn5.;` | `18II` |
| `put date yyqrp8.;` | `2018.II` |
| `put date yyqrs10.;` | `2018/II` |

## See Also

**Formats:**

- "YYQxw." on page 508
- "YYQRw." on page 510

# YYQZw. Format

Writes SAS date values in the form <*yy*><*qq*>, the year appears as 2 or 4 digits, and *qq* is the quarter of the year.

Categories: CAS
Date and Time

Alignment: Right

## Syntax

**YYQZ***w*.

## Required Arguments

***Z***
specifies that no separator appear between the year and the quarter.

***w***
specifies the width of the output field.

**Default** 4

**Range** 4–6

## Details

The YYQZ*w*. format writes SAS date values in the form <*yy*> <*qq*>. Here is an explanation of the syntax:

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

Z
specifies that there is no separator.

<*qq*>
is an integer (01, 02, 03, or 04) that represents the quarter of the year.

## Example

The following examples use the input value of 20999. 20999 is the SAS date value that corresponds to June 29, 2017.

```
data _null_;
   declare char(20) a b;
   method run();
      a=put (20999,yyqz6.);
      b=put (20999,yyqz4.);
      put a= b=;
   end;
enddata;
run;
```

SAS writes the following output to the log.

```
a=201702              b=1702
```

## See Also

- "DS2 Expressions" in *SAS DS2 Programmer's Guide*

**Formats:**

- "YYQw." in *SAS DS2 Language Reference*
- "YYQxw." in *SAS DS2 Language Reference*
- "YYQRw." in *SAS DS2 Language Reference*
- "YYQRxw." in *SAS DS2 Language Reference*

# YYWEEKUw. Format

Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.

Categories: CAS
Date and Time

Alignment: Left

# Syntax

**YYWEEKU***w*.

## Syntax Description

***w***
specifies the width of the output field.

**Default** 7

**Range** 2-8

# Details

The YYWEEKU*w*. format writes a week-number format. The YYWEEKU*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).

For more information about widths, formats, and examples see the following table:

| Widths | Formats | Examples |
|---|---|---|
| 3-4 | Www | W01 |
| 5-6 | yyWww | 07W01 |
| 7 | yyyyWww | 2007W01 |
| 8 | yyyy-Www | 2007-W01 |
| 9-above | invalid | invalid |

# Comparisons

The YYWEEKU*w*. format is similar to the WEEKU*w*. format except that the YYWEEKU*w*. format does not specify the day-of-week information. Also, the YYWEEKU*w*. format does not accept any width that is greater than 8.

## Example

```
sasdate = '01JAN2007'd;
```

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `u=put(sasdate,yyweeku3.);`<br>`v=put(sasdate,yyweeku4.);`<br>`w=put(sasdate,yyweeku5.);`<br>`x=put(sasdate,yyweeku6.);`<br>`y=put(sasdate,yyweeku7.);`<br>`z=put(sasdate,yyweeku8.);`<br>`put u;`<br>`put v;`<br>`put w;`<br>`put x;`<br>`put y;`<br>`put z;` | `W00`<br>`W00`<br>`07W00`<br>`07W00`<br>`2007W00`<br>`2007-W00` |

## See Also

**Format:**

- "WEEKUw." in *SAS National Language Support (NLS): Reference Guide*

# YYWEEKVw. Format

Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**YYWEEKV***w*.

## Syntax Description

***w***
specifies the width of the output field.

**Default** 7

**Range** 2–8

# Details

The YYWEEKV*w*. format writes the various formats depending on the specified width. Algorithm V calculates the SAS date value, and the number-of-the-week value is represented as a decimal number in the range 01–53, with a leading 0 and maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 4 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.

For more information about widths, formats, and examples see the following table:

| Widths | Formats | Examples |
|---|---|---|
| 3-4 | www | w01 |
| 5-6 | yyWww | 07W01 |
| 7 | yyyyWww | 2007W01 |
| 8 | yyyy-Www | 2007-W01 |
| 9-above | invalid | invalid |

# Comparisons

The YYWEEKV*w*. format is similar to the WEEKV*w*. format except that the YYWEEKV*w*. format does not specify the day-of-week information. Also, the YYWEEKV*w*. format does not accept a width that is greater than 8.

# Example

```
sasdate = '01JAN2007'd;
```

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `u=put(sasdate,yyweekv3.);`<br>`v=put(sasdate,yyweekv4.);` | `W01`<br>`W01` |

| Statements | Results |
|---|---|
| `w=put(sasdate,yyweekv5.);` | `07W01` |
| `x=put(sasdate,yyweekv6.);` | `07W01` |
| `y=put(sasdate,yyweekv7.);` | `2007W01` |
| `z=put(sasdate,yyweekv8.);` | `2007-W01` |
| `put u;` | |
| `put v;` | |
| `put w;` | |
| `put x;` | |
| `put y;` | |
| `put z;` | |

## See Also

**Format:**

- "WEEKVw." in *SAS National Language Support (NLS): Reference Guide*

# YYWEEKWw. Format

Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.

Categories: CAS
Date and Time

Alignment: Left

## Syntax

**YYWEEKW***w*.

## Syntax Description

***w***

specifies the width of the output field.

**Default** 7

**Range** 2–8

# Details

The YYWEEKW*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.

For more information about widths, formats, and examples see the following table:

| Widths | Formats | Examples |
|---|---|---|
| 3-4 | Www | W01 |
| 5-6 | yyWww | 07W01 |
| 7 | yyyyWww | 2007W01 |
| 8 | yyyy-Www | 2007-W01 |
| 9-above | invalid | invalid |

# Comparisons

The YYWEEKW*w*. format is similar to the WEEKW*w*. format except that the YYWEEKW*w*. format does not specify the day-of-week information. Also, the YYWEEKW*w*. format does not accept any width that is greater than 8.

# Example

```
sasdate = '01JAN2007'd
```

| Statements | Results |
|---|---|
| | `----+----1----+` |
| `u=put(sasdate,yyweekw3.);`<br>`v=put(sasdate,yyweekw4.);`<br>`w=put(sasdate,yyweekw5.);`<br>`x=put(sasdate,yyweekw6.);`<br>`y=put(sasdate,yyweekw7.);`<br>`z=put(sasdate,yyweekw8.);`<br>`put u;`<br>`put v;`<br>`put w;`<br>`put x;`<br>`put y;`<br>`put z;` | `W01`<br>`W01`<br>`07W01`<br>`07W01`<br>`2007W01`<br>`2007-W01` |

## See Also

**Format:**

- "WEEKWw." in *SAS National Language Support (NLS): Reference Guide*

# Zw.d Format

Writes standard numeric data with leading 0s.

Categories: CAS
Numeric

Alignment: Right

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

## 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*, *Zw.d* writes the value without a decimal point.

## Details

The *Zw.d* format writes standard numeric values one digit per byte and fills in 0s to the left of the data value.

The *Zw.d* format rounds to the nearest number that fits in the output field. If *w.d* is too large to fit, SAS might shift the decimal to the BEST*w*. format. The *Zw.d* format writes negative numbers with leading minus signs. In addition, it right-aligns before writing and pads the output with leading 0s.

## Comparisons

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

## Example

```
data one;
   num=1350;
   put num z8.;
run;
```

| Value of num | Result |
|---|---|
| | `----+----1` |
| 1350 | `00001350` |

# ZDw.d Format

Writes numeric data in zoned decimal format.

Category: Numeric

Alignment: Left

Restriction: This format is not supported in a DATA step that runs in CAS.

Interaction: When the DECIMALCONV= system option is set to STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see "DECIMALCONV=" in *SAS System Options: Reference*.

See: "ZD Format: UNIX" in *SAS Companion for UNIX Environments*

"ZDw.d: 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 1 byte. However, the value's sign is in the last byte, along with the last digit.

Delimiters are not supported in the PUT statement's LIST output.

**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

```
data one;
   num=1350;
   put num zd8.;
run;
```

| Value of num | Result [1] |
|---|---|
| 1350 | 0000135{ |

1 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

# 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. "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 8-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 *d* values from the informat, SAS uses default values. If the data contains decimal points, SAS ignores the *d* value and reads the number of decimal places that are actually in the input data.

If the informat width is too narrow to read all the columns in the input data, you 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 "Definitions for Dates, Times, and Intervals" on page 9.

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 these 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" in *SAS DATA Step 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. Here is an example:

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

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.

> **TIP** If the first argument of the INPUT function is a character expression and not a variable, the character expression must include quotation marks.

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" in *SAS Functions and CALL Routines: Reference*. For a complete discussion of the INPUT function, see "INPUT" 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 information about using the colon (:) modifier, see "INPUT: List" in *SAS DATA Step Statements: Reference*. Therefore, SAS uses a modified list input to read the variable and ensure that these rules are followed:

- 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.

For more information about how to use modified list input to read data, see "INPUT: List" in *SAS DATA Step Statements: Reference* .

**Note:** Anytime a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option in either the ASCII or EBCDIC environment. 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 (such as quotation marks, commas, and blanks) in the session encoding. The use of encoding-specific informats should be reserved for use with true binary files (that is, binary files that 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: List" in *SAS DATA Step 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" in *SAS DATA Step 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. 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 "FORMAT Procedure" in *Base SAS Procedures Guide*.

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

- create permanent, not temporary, informats with PROC FORMAT
- 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, the following system options produce these results:

| System Option | 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" 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: 1 byte, 2 bytes, or 4 bytes. 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 platforms that are based on Intel (little endian). In SAS, 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, 2 bytes as 00 01, and 4 bytes as 00 00 00 01. On little endian platforms, the value 1 is stored in 1 byte as 01 (the same as big endian), in 2 bytes as 01 00, and in 4 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 is set to 0. For example, –2 would be represented in 1, 2, and 4 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 and Little Endian Platforms

SAS can read signed and unsigned integers regardless of whether they were generated on a big endian or a little endian platform. 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 8 bytes.

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

| 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 [1] |
| Java | short | int | long [1] |
| Visual Basic 6.0 | short | long* | none |
| Visual Basic.NET | short | integer | long [1] |
| 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 |

1 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 1 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, the number is negative. Otherwise, it is positive.

The following information 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 handle the representation. It is consistent between ASCII and EBCDIC platforms.
- Note that the S370FPDU format and informat expect to have an F in the last nibble, whereas packed decimal data expects no sign nibble.

## Zoned Decimal Data

The following information 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 that contains 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 expect 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 2 bytes. The first byte contains the hexadecimal value F1 and the second byte contains the hexadecimal value C5.

## Packed Julian Dates

The following information 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, whereas 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 two-digit or four-digit years. If you use two-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. Other platforms must use software to emulate the computations. For example, the

IBM mainframe has an Add Pack instruction to add packed decimal data, but the platforms that are based on Intel have no such instruction and must convert the decimal data to 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 in the preceding table, 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 3 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 data, and the Z directive indicates zoned decimal data. Here is an excerpt from an assembly language list that shows 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 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 | Comment |
|---|---|---|---|
| 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 | Zoned decimal representation other than IBM. |
| 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' |

| Format | Data Type Representation | Corresponding Informat | Comment |
|---|---|---|---|
| | | | (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'). |
| PDJULI | Packed decimal | PDJULI | Julian date in packed representation - IBM computation |
| PDJULG | Packed decimal | PDJULG | Julian date in packed representation - Gregorian computation |
| none | Packed decimal | RMFDUR | Input layout is: *mmssttt*F |
| none | Packed decimal | SHRSTAMP | Input layout is *yyyyddd*F*hhmmssth*, where *yyyyddd*F is the packed Julian date; *yyyy* is a zero-based year from 1900. |
| none | Packed decimal | SMFSTAMP | Input layout is *xxxxxxxxyyyyddd*F, where *yyyyddd*F is the packed Julian date; *yyyy* is a zero-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 zero-based year from 1900. |

# Reading 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 values of the ISO 8601 dates, time, datetime, durations, and interval:

***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* uses 1–3 digits for values read by the $N8601B informat and the $N8601E informat.

*ffffff* use 1–6 digits for informats other than the $N8601B and $N8601E informats.

**Y**
indicates that a year value precedes this character in a duration.

**M**
as part of a date, indicates that a month value precedes this character in a duration.

**D**
indicates that a day value precedes this character in a duration.

**H**
indicates that an hour value precedes this character in a duration.

**M**
as part of a time, indicates that a minute value precedes this character in a duration.

**S**
indicates that a seconds value precedes 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 | 20180915 | B8601DA*w*. |
| Time with no time zone offset | *hhmmssnnnnnn* | 155300322348 | B8601TM*w.d* |
| Time with a time zone offset | *hhmmss*+\|–*hhmm* | 155300–0500 | B8601TZ*w.d* |
| | *hhmmss*Z | 155300Z | B8601TZ*w.d* |
| Convert to local time with a time zone offset | *hhmmss*+\|–*hhmm* | 155300+0500 | B8601TZ*w.d* |

| Date, Time, or Datetime | ISO 8601 Notation | Example | Informat |
|---|---|---|---|
| Datetime with no time zone offset | *yyyymmdd*T*hhmmssnnnnnn* | 20180915T155300 | B8601DT*w.d* |
| Datetime with a time zone offset | *yyyymmdd*T*hhmmss*+\|–*hhmm* | 20180915T155300+0500 | B8601DZ*w.d* |
| | *yyyymmdd*T*hhmmss*Z | 20180915T155300Z | B8601DZ*w.d* |
| Date from a datetime and no time zone offset | *yyyymmdd* | 20180915 | B8601DN*w*. |
| Extended Notations | | | |
| Date | *yyyy-mm-dd* | 2018-09-15 | E8601DA*w*. |
| Time with no time zone offset | *hh:mm:ss.nnnnnn* | 15:53:00.322348 | E8601TM*w.d* |
| Time with a time zone offset | hh:mm:ss.nnnnnn+\|–hh:mm | 15:53:00+05:00 | E8601TZ*w.d* |
| Convert to local time with a time zone offset | *hh:mm:ss.nnnnnn*+\|–*hh:mm* | 15:53:00+05:00 | E8601LZ*w.d* |
| Datetime with no time zone offset | *yyyy-mm-dd*T*hh:mm:ss.nnnnnn* | 2018-09-15T15:53:00 | E8601DT*w.d* |
| Datetime with a time zone offset | *yyyy-mm-dd*T*hh:mm:ss.nnnnnn*+\|–*hh:mm* | 2018-09-15T15:53:00+05:00 | E8601DZ*w.d* |
| Date from a datetime and no time zone offset | *yyyy-mm-dd* | 2018-09-15 | 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 2018-09-15T15:53:00+05:00 using the E8601DZ informat, the datetime value 1852645980 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 2018–09–15T10:53:00+00:00. The hour specified after the T shows the five-hour adjustment.

For examples of reading ISO 8601 dates, times, and datetimes, see "Examples of Reading and Writing Basic and Extended ISO 8601 Date, Time, and Datetime Values" on page 51.

# Reading ISO 8601 Duration, Interval, and Datetime Values

## Informats That Read Duration, Interval, and Datetime Values

SAS uses two informats that read 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:

| Time Component | ISO 8601 Notation | Example |
|---|---|---|
| Duration - Basic Notation | PYYYYMMDDThhmmss | P20180915T155300 |
| | –PYYYYMMDDThhmmss | –P20180915T155300 |
| Duration - Extended Notation | PYYYY-MM-DDThh:mm:ss | P2018-09-15T15:53:00 |
| | –PYYYY-MM-DDThh:mm:ss | –P2018-09-15T15:53:00 |

| Time Component | ISO 8601 Notation | Example |
|---|---|---|
| Duration - Basic and Extended Notation | PnYnMnDTnHnMnS | P2y10m14dT20h13m45s |
| | –PnYnMnDTnHnMnS | –P2n10m14dT20h13m45s |
| | PnW (weeks) | P6w |
| Interval - Basic Notation | YYYYMMDDThhmmss/ YYYYMMDDThhmmss | 20180915T155300/201411 13T000000 |
| | PnYnMnDTnHnMnS/ YYYYMMDDThhmmss | P2y10M14dT20h13m45s/ 20180915T155300 |
| | YYYYMMDDThhmmss/ PnYnMnDTnHnMnS | 20180915T155300/ P2y10M14dT20h13m45s |
| Interval - Extended Notation | YYYY-MM-DDThh:mm:ss/ YYYY-MM-DDThh:mm:ss | 2018-09-15T15:53:00/201 4-11-13T00:00:00 |
| | PnYnMnDTnHnMnS/ YYYY-MM-DDThh:mm:ss | P2y10M14dT20h13m45s/ 2018-09-15T15:53:00 |
| | YYYY-MM-DDThh:mm:ss/ PnYnMnDTnHnMnS | 2018-09-15T15:53:00/ P2y10M14dT20h13m45s |
| Datetime-Basic Notation | YYYYMMDDThhmmss.fff +\|–hhmm | 20180915T155300 |
| | (all blank) | |
| Datetime-Extended Notation | YYYY-MM-DDThh:mm:ss.fff+\|–hhmm | 2018-09-15T15:53:00 +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 by 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.

`2018-09-15T15:x:00/2016-09-15T15: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`
- `2016-09-15T15/2018-09-15T15:53`
- `-p0003-03-03T-:-:-`
- `P2y3m4dT5h6m`
- `2018-09-xTx:x:x`
- `2018`

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 Duration, Datetime, and Interval Values

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

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

# 4

# Dictionary of Informats

# Informats Documented in Other Publications

Additional informats are documented in National Language Support (NLS): Reference Guide:"Dictionary of Informats for NLS" in *SAS National Language Support (NLS): Reference Guide*..

# Informats by Category

Here are the categories for informats:

| Category | Description |
| --- | --- |
| CAS | instructs SAS to read character data values into character variables in CAS.<br>See CAS for a list of informats. |
| Character | instructs SAS to read character data values into character variables. |

| Category | Description |
|---|---|
| | See Character for a list of informats. |
| Column Binary | instructs SAS to read data stored in column-binary or multipunched form into character or numeric variables.<br>See Column Binary for a list of informats. |
| Date | Instructs SAS to write data values from variables that represent dates.<br>See Date for a list of informats. |
| Date and Time | instructs SAS to read date values into variables that represent dates, times, and datetimes.<br>See Date and Time for a list of informats. |
| 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.<br>See ISO 8601 for a list of informats. |
| Numeric | instructs SAS to read numeric data values into numeric variables.<br>See Numeric for a list of informats. |
| Time | Instructs SAS to write data values from variables that represent times.<br>See Time for a list of informats. |

For information about column-binary data, see "Description of Column-Binary Data Storage" in *SAS Programmer's Guide: Essentials*. For information about creating user-defined informats, see "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 |
|---|---|---|
| CAS | $ASCIIw. Informat (p. 562) | Converts ASCII character data to native format. |
| | $BASE64Xw. Informat (p. 563) | Converts ASCII text to character data by using Base 64 encoding. |
| | $BINARYw. Informat (p. 564) | Converts binary data to character data. |
| | $CBw. Informat (p. 566) | Reads standard character data from column-binary files. |
| | $CHARw. Informat (p. 567) | Reads character data with blanks. |

| Category | Language Elements | Description |
|---|---|---|
| | $CHARZBw. Informat (p. 569) | Converts binary 0s to blanks. |
| | $EBCDICw. Informat (p. 570) | Converts EBCDIC character data to native format. |
| | $HEXw. Informat (p. 571) | Converts hexadecimal data to character data. |
| | $N8601Bw.d Informat (p. 573) | Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notation. |
| | $N8601Ew.d Informat (p. 575) | Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation. |
| | $OCTALw. Informat (p. 578) | Converts octal data to character data. |
| | $PHEXw. Informat (p. 579) | Converts packed hexadecimal data to character data. |
| | $QUOTEw. Informat (p. 580) | Removes matching quotation marks from character data. |
| | $UPCASEw. Informat (p. 581) | Converts character data to uppercase. |
| | $VARYINGw. Informat (p. 583) | Reads character data of varying length. |
| | $w. Informat (p. 585) | Reads standard character data. |
| | ANYDTDTEw. Informat (p. 586) | Reads and extracts the date value from various date, time, and datetime forms. |
| | ANYDTDTMw. Informat (p. 589) | Reads and extracts datetime values from various date, time, and datetime forms. |
| | ANYDTTMEw. Informat (p. 592) | Reads and extracts time values from various date, time, and datetime forms. |
| | BINARYw.d Informat (p. 609) | Converts positive binary values to integers. |
| | CBw.d Informat (p. 613) | Reads standard numeric values from column-binary files. |
| | COMMAw.d Informat (p. 614) | Removes embedded characters. |
| | COMMAXw.d Informat (p. 615) | Removes embedded periods, blanks, dollar signs, percent signs, hyphens, and close parentheses 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. |
| | DATEw. Informat (p. 617) | Reads date values in the form ddmmmyy or ddmmmyyyy. |

| Category | Language Elements | Description |
|---|---|---|
| | DATETIMEw. Informat (p. 618) | Reads datetime values in the form ddmmmyy hh:mm:ss.ss or ddmmmyyyy hh:mm:ss.ss. |
| | DDMMYYw. Informat (p. 621) | 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 two or four digits. |
| | E8601DAw. Informat (p. 622) | Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DNw. Informat (p. 624) | 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. 625) | Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.<ffffff>. |
| | E8601DZw.d Informat (p. 627) | 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. |
| | E8601TMw.d Informat (p. 632) | Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.<ffffff>. |
| | E8601TZw.d Informat (p. 633) | Reads time values that are specified using the ISO 8601 extended time notation hh:mm:ss+\|–hh:mm.<ffffff> or hh:mm:ssZ. |
| | FLOATw.d Informat (p. 635) | Reads a native single-precision, floating-point value and divides it by 10 raised to the dth power. |
| | HEXw. Informat (p. 637) | Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values. |
| | HHMMSSw. Informat (p. 638) | Reads hours, minutes, and seconds in the form hh:mm:ss or hhmmss. |
| | IBw.d Informat (p. 640) | Reads native integer binary (fixed-point) values, including negative values. |
| | IBRw.d Informat (p. 642) | Reads integer binary (fixed-point) values in Intel and DEC formats. |
| | IEEEw.d Informat (p. 644) | Reads an IEEE floating-point value and divides it by 10 raised to the dth power. |
| | JULIANw. Informat (p. 645) | Reads Julian dates in the form yyddd or yyyyddd. |
| | MDYAMPMw.d Informat (p. 647) | 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 two or four digits. |

| Category | Language Elements | Description |
|---|---|---|
| | MMDDYYw. Informat (p. 649) | Reads date values in the form mmddyy or mmddyyyy. |
| | MONYYw. Informat (p. 651) | Reads month and year date values in the form mmmyy or mmmyyyy. |
| | MSECw. Informat (p. 653) | Reads TIME MIC values. |
| | NUMXw.d Informat (p. 654) | Reads numeric values with a comma in place of the decimal point. |
| | PDw.d Informat (p. 656) | Reads data that is stored in IBM packed decimal format. |
| | PDTIMEw. Informat (p. 661) | Reads packed decimal time of SMF and RMF records. |
| | PERCENTw.d Informat (p. 663) | Reads percentages as numeric values. |
| | PIBw.d Informat (p. 664) | Reads positive integer binary (fixed-point) values. |
| | PIBRw.d Informat (p. 666) | Reads positive integer binary (fixed-point) values in Intel and DEC formats. |
| | PKw.d Informat (p. 668) | Reads unsigned packed decimal data. |
| | PUNCH.d Informat (p. 669) | Reads whether a row of column-binary data is punched. |
| | RBw.d Informat (p. 670) | Reads numeric data that is stored in real binary (floating-point) notation. |
| | RMFDURw. Informat (p. 672) | Reads duration intervals of RMF records. |
| | RMFSTAMPw. Informat (p. 674) | Reads time and date fields of RMF records. |
| | S370FFw.d Informat (p. 677) | Reads EBCDIC numeric data. |
| | S370FIBw.d Informat (p. 679) | Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format. |
| | S370FIBUw.d Informat (p. 680) | Reads unsigned integer binary (fixed-point) values in IBM mainframe format. |
| | S370FPDw.d Informat (p. 682) | Reads packed data in IBM mainframe format. |
| | S370FPDUw.d Informat (p. 684) | Reads unsigned packed decimal data in IBM mainframe format. |
| | S370FPIBw.d Informat (p. 685) | Reads positive integer binary (fixed-point) values in IBM mainframe format. |

| Category | Language Elements | Description |
|---|---|---|
| | S370FRBw.d Informat (p. 687) | Reads real binary (floating-point) data in IBM mainframe format. |
| | S370FZDw.d Informat (p. 689) | Reads zoned decimal data in IBM mainframe format. |
| | S370FZDBw.d Informat (p. 690) | Reads zoned decimal data in which 0s have been left blank. |
| | S370FZDLw.d Informat (p. 692) | Reads zoned decimal leading-sign data in IBM mainframe format. |
| | S370FZDSw.d Informat (p. 693) | Reads zoned decimal, separate leading-sign data in IBM mainframe format. |
| | S370FZDTw.d Informat (p. 694) | Reads zoned decimal, separate trailing-sign data in IBM mainframe format. |
| | S370FZDUw.d Informat (p. 696) | Reads unsigned zoned decimal data in IBM mainframe format. |
| | SHRSTAMPw. Informat (p. 697) | Reads date and time values of SHR records. |
| | SMFSTAMPw. Informat (p. 699) | Reads time and date values of SMF records. |
| | TIMEw. Informat (p. 702) | 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. 704) | Reads an 8-byte time-of-day stamp. |
| | TUw. Informat (p. 706) | Reads timer units. |
| | VAXRBw.d Informat (p. 707) | Reads real binary (floating-point) data in VMS format. |
| | VMSZNw.d Informat (p. 708) | Reads VMS and MicroFocus COBOL zoned numeric data. |
| | w.d Informat (p. 710) | Reads standard numeric data. |
| | YYMMDDw. Informat (p. 720) | Reads date values in the form yymmdd or yyyymmdd. |
| | YYMMNw. Informat (p. 722) | Reads date values in the form yyyymm or yymm. |
| | YYQw. Informat (p. 724) | Reads quarters of the year in the form yyQq or yyyyQq. |
| | ZDw.d Informat (p. 725) | Reads zoned decimal data. |

| Category | Language Elements | Description |
|---|---|---|
| Character | $ASCIIw. Informat (p. 562) | Converts ASCII character data to native format. |
| | $BASE64Xw. Informat (p. 563) | Converts ASCII text to character data by using Base 64 encoding. |
| | $BINARYw. Informat (p. 564) | Converts binary data to character data. |
| | $CHARw. Informat (p. 567) | Reads character data with blanks. |
| | $CHARZBw. Informat (p. 569) | Converts binary 0s to blanks. |
| | $EBCDICw. Informat (p. 570) | Converts EBCDIC character data to native format. |
| | $HEXw. Informat (p. 571) | Converts hexadecimal data to character data. |
| | $OCTALw. Informat (p. 578) | Converts octal data to character data. |
| | $PHEXw. Informat (p. 579) | Converts packed hexadecimal data to character data. |
| | $QUOTEw. Informat (p. 580) | Removes matching quotation marks from character data. |
| | $UPCASEw. Informat (p. 581) | Converts character data to uppercase. |
| | $VARYINGw. Informat (p. 583) | Reads character data of varying length. |
| | $w. Informat (p. 585) | Reads standard character data. |
| Column Binary | $CBw. Informat (p. 566) | Reads standard character data from column-binary files. |
| | CBw.d Informat (p. 613) | Reads standard numeric values from column-binary files. |
| | PUNCH.d Informat (p. 669) | Reads whether a row of column-binary data is punched. |
| | ROWw.d Informat (p. 675) | Reads a column-binary field down a card column. |
| Date | B8601DAw. Informat (p. 596) | Reads date values that are specified using the ISO 8601 base notation yyyymmdd. |
| | B8601DNw. Informat (p. 599) | 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. |
| | DATEw. Informat (p. 617) | Reads date values in the form ddmmmyy or ddmmmyyyy. |
| | DDMMYYw. Informat (p. 621) | 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 two or four digits. |

| Category | Language Elements | Description |
|---|---|---|
| | E8601DAw. Informat (p. 622) | Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DNw. Informat (p. 624) | 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. |
| | MMDDYYw. Informat (p. 649) | Reads date values in the form mmddyy or mmddyyyy. |
| | MONYYw. Informat (p. 651) | Reads month and year date values in the form mmmyy or mmmyyyy. |
| | PDJULGw. Informat (p. 658) | Reads packed Julian date values in the hexadecimal form yyyydddF for IBM. |
| | PDJULIw. Informat (p. 660) | Reads packed Julian dates in the hexadecimal format ccyydddF for IBM. |
| | WEEKUw. Informat (p. 711) | 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. 713) | Reads a value in the form of a week number within a year and returns a SAS date value by using the V algorithm. |
| | WEEKWw. Informat (p. 716) | Reads a value in the form of a week number within the year and returns a SAS date value by using the W algorithm. |
| | YYMMDDw. Informat (p. 720) | Reads date values in the form yymmdd or yyyymmdd. |
| | YYMMNw. Informat (p. 722) | Reads date values in the form yyyymm or yymm. |
| | YYQw. Informat (p. 724) | Reads quarters of the year in the form yyQq or yyyyQq. |
| Date and Time | $N8601Bw.d Informat (p. 573) | Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notation. |
| | $N8601Ew.d Informat (p. 575) | Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation. |
| | ANYDTDTEw. Informat (p. 586) | Reads and extracts the date value from various date, time, and datetime forms. |
| | ANYDTDTMw. Informat (p. 589) | Reads and extracts datetime values from various date, time, and datetime forms. |
| | ANYDTTMEw. Informat (p. 592) | Reads and extracts time values from various date, time, and datetime forms. |
| | B8601CIw.d Informat (p. 594) | Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss<fff>. |

| Category | Language Elements | Description |
|---|---|---|
| | B8601DJw.d Informat (p. 598) | Reads a Java date and time value that is in the form yyyymmddhhmmss<ffffff>. |
| | B8601DNw. Informat (p. 599) | 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. 601) | Reads datetime values that are specified using the ISO 8601 basic notation yyyymmddThhmmss<ffffff>. |
| | B8601DZw.d Informat (p. 603) | Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation yyyymmddThhmmss+\|–hhmm or yyyymmddThhmmss<ffffff>Z. |
| | DATETIMEw. Informat (p. 618) | Reads datetime values in the form ddmmmyy hh:mm:ss.ss or ddmmmyyyy hh:mm:ss.ss. |
| | E8601DTw.d Informat (p. 625) | Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.<ffffff>. |
| | E8601DZw.d Informat (p. 627) | 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. |
| | JULIANw. Informat (p. 645) | Reads Julian dates in the form yyddd or yyyyddd. |
| | MDYAMPMw.d Informat (p. 647) | 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 two or four digits. |
| | RMFDURw. Informat (p. 672) | Reads duration intervals of RMF records. |
| | RMFSTAMPw. Informat (p. 674) | Reads time and date fields of RMF records. |
| | SHRSTAMPw. Informat (p. 697) | Reads date and time values of SHR records. |
| | SMFSTAMPw. Informat (p. 699) | Reads time and date values of SMF records. |
| | YMDDTTMw.d Informat (p. 718) | 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 two or four digits. |
| ISO 8601 | $N8601Bw.d Informat (p. 573) | Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notation. |

| Category | Language Elements | Description |
|---|---|---|
| | $N8601Ew.d Informat (p. 575) | Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation. |
| | B8601CIw.d Informat (p. 594) | Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss<fff>. |
| | B8601DAw. Informat (p. 596) | Reads date values that are specified using the ISO 8601 base notation yyyymmdd. |
| | B8601DJw.d Informat (p. 598) | Reads a Java date and time value that is in the form yyyymmddhhmmss<ffffff>. |
| | B8601DNw. Informat (p. 599) | 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. 601) | Reads datetime values that are specified using the ISO 8601 basic notation yyyymmddThhmmss<ffffff>. |
| | B8601DZw.d Informat (p. 603) | 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. 605) | Reads time values that are specified using the ISO 8601 basic notation hhmmss<ffffff>. |
| | B8601TZw.d Informat (p. 607) | Reads time values that are specified using the ISO 8601 basic time notation hhmmss<ffff>+\|–hhmm or hhmmss<ffffff>Z. |
| | E8601DAw. Informat (p. 622) | Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd. |
| | E8601DNw. Informat (p. 624) | 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. 625) | Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.<ffffff>. |
| | E8601DZw.d Informat (p. 627) | 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. 630) | 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. 632) | Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.<ffffff>. |

| Category | Language Elements | Description |
|---|---|---|
| | E8601TZw.d Informat (p. 633) | 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. 609) | Converts positive binary values to integers. |
| | BITSw.d Informat (p. 610) | Extracts bits. |
| | BZw.d Informat (p. 611) | Converts blanks to 0s. |
| | COMMAw.d Informat (p. 614) | Removes embedded characters. |
| | COMMAXw.d Informat (p. 615) | Removes embedded periods, blanks, dollar signs, percent signs, hyphens, and close parentheses 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. |
| | FLOATw.d Informat (p. 635) | Reads a native single-precision, floating-point value and divides it by 10 raised to the dth power. |
| | HEXw. Informat (p. 637) | Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values. |
| | IBw.d Informat (p. 640) | Reads native integer binary (fixed-point) values, including negative values. |
| | IBRw.d Informat (p. 642) | Reads integer binary (fixed-point) values in Intel and DEC formats. |
| | IEEEw.d Informat (p. 644) | Reads an IEEE floating-point value and divides it by 10 raised to the dth power. |
| | NUMXw.d Informat (p. 654) | Reads numeric values with a comma in place of the decimal point. |
| | OCTALw.d Informat (p. 655) | Converts positive octal values to integers. |
| | PDw.d Informat (p. 656) | Reads data that is stored in IBM packed decimal format. |
| | PERCENTw.d Informat (p. 663) | Reads percentages as numeric values. |
| | PIBw.d Informat (p. 664) | Reads positive integer binary (fixed-point) values. |
| | PIBRw.d Informat (p. 666) | Reads positive integer binary (fixed-point) values in Intel and DEC formats. |
| | PKw.d Informat (p. 668) | Reads unsigned packed decimal data. |
| | RBw.d Informat (p. 670) | Reads numeric data that is stored in real binary (floating-point) notation. |

| Category | Language Elements | Description |
|---|---|---|
| | S370FFw.d Informat (p. 677) | Reads EBCDIC numeric data. |
| | S370FIBw.d Informat (p. 679) | Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format. |
| | S370FIBUw.d Informat (p. 680) | Reads unsigned integer binary (fixed-point) values in IBM mainframe format. |
| | S370FPDw.d Informat (p. 682) | Reads packed data in IBM mainframe format. |
| | S370FPDUw.d Informat (p. 684) | Reads unsigned packed decimal data in IBM mainframe format. |
| | S370FPIBw.d Informat (p. 685) | Reads positive integer binary (fixed-point) values in IBM mainframe format. |
| | S370FRBw.d Informat (p. 687) | Reads real binary (floating-point) data in IBM mainframe format. |
| | S370FZDw.d Informat (p. 689) | Reads zoned decimal data in IBM mainframe format. |
| | S370FZDBw.d Informat (p. 690) | Reads zoned decimal data in which 0s have been left blank. |
| | S370FZDLw.d Informat (p. 692) | Reads zoned decimal leading-sign data in IBM mainframe format. |
| | S370FZDSw.d Informat (p. 693) | Reads zoned decimal, separate leading-sign data in IBM mainframe format. |
| | S370FZDTw.d Informat (p. 694) | Reads zoned decimal, separate trailing-sign data in IBM mainframe format. |
| | S370FZDUw.d Informat (p. 696) | Reads unsigned zoned decimal data in IBM mainframe format. |
| | TRAILSGNw. Informat (p. 705) | Reads a trailing plus (+) or minus (–) sign. |
| | VAXRBw.d Informat (p. 707) | Reads real binary (floating-point) data in VMS format. |
| | VMSZNw.d Informat (p. 708) | Reads VMS and MicroFocus COBOL zoned numeric data. |
| | w.d Informat (p. 710) | Reads standard numeric data. |
| | ZDw.d Informat (p. 725) | Reads zoned decimal data. |
| | ZDBw.d Informat (p. 727) | Reads zoned decimal data in which zeros have been left blank. |

| Category | Language Elements | Description |
|---|---|---|
| | ZDVw.d Informat (p. 728) | Reads and validates zoned decimal data. |
| Time | B8601TMw.d Informat (p. 605) | Reads time values that are specified using the ISO 8601 basic notation hhmmss<ffffff>. |
| | B8601TZw.d Informat (p. 607) | Reads time values that are specified using the ISO 8601 basic time notation hhmmss<ffff>+\|–hhmm or hhmmss<ffffff>Z. |
| | E8601LZw.d Informat (p. 630) | 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. 632) | Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.<ffffff>. |
| | E8601TZw.d Informat (p. 633) | 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. 638) | Reads hours, minutes, and seconds in the form hh:mm:ss or hhmmss. |
| | MSECw. Informat (p. 653) | Reads TIME MIC values. |
| | PDTIMEw. Informat (p. 661) | Reads packed decimal time of SMF and RMF records. |
| | STIMERw. Informat (p. 700) | Reads time values and determines whether the values are hours, minutes, or seconds; reads the output of the STIMER system option. |
| | TIMEw. Informat (p. 702) | 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. 704) | Reads an 8-byte time-of-day stamp. |
| | TUw. Informat (p. 706) | Reads timer units. |

# Dictionary

## $ASCIIw. Informat

Converts ASCII character data to native format.

Categories: CAS
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

```
data one;
   input name $ascii3.;
   put name $hex.;
datalines;
abc
ABC
;
run;
```

*Table 4.1* *Output on EBCDIC and ASCII Systems*

| Variable | EBCDIC Results | ASCII Results |
|---|---|---|
| abc | 818283 | 818283 |
| ABC | C1C2C3 | 414243 |

**Note:** The results in this table is hexadecimal representations of codes for characters. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one character value.

# $BASE64Xw. Informat

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

Categories: CAS
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 email 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 /.

Here 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

```
data _null_;
    B64EXMPL="RkNBMDFBNzk5M0JD";
    B64EXMPL2=input(B64EXMPL, $base64x64.);
    put B64EXMPL / B64EXMPL2 ;
 run;
```

| Variable | Result |
|---|---|
| B64EXMPL | RkNBMDFBNzk5M0JD |
| B64EXMPL2 | FCA01A7993BC |

## See Also

- The XMLDOUBLE option of the "LIBNAME Statement: XMLV2 and XML Engines" in *SAS XMLV2 and XML LIBNAME Engines: User's Guide*

**Formats:**

- "$BASE64Xw." on page 104

# $BINARYw. Informat

Converts binary data to character data.

Categories: CAS
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 become 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 1 byte of numeric data.
- The $HEX*w*. informat reads hexadecimal characters that represent the ASCII or EBCDIC equivalent of character data.

# Example

```
data one;
```

```
   name='0100110001001101';
   name2=input(name,$binary16.);

   put name ;
   put name2 ;
run;
```

| Variable | Result |
|---|---|
| B64EXMPL | RkNBMDFBNzk5M0JD |
| put name2; | LM |

# $CBw. Informat

Reads standard character data from column-binary files.

Categories: CAS
Column Binary

## Syntax

**$CB***w*.

## Syntax Description

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

**Default** 1

**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, where each card column is represented in 2 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

The variable 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.

```
data _null_;
   a=input('200a'x,$cb2.);
   put a=;
run;
```

***Table 4.2*** *Output on EBCDIC and ASCII Systems*

| Variable | EBCDIC Results | ASCII Results |
|---|---|---|
| a | + | N |

## See Also

- “Description of Column-Binary Data Storage” in *SAS Programmer’s Guide: Essentials*

**Informats:**

- “CBw.d” on page 613
- “PUNCH.d” on page 669
- “ROWw.d” on page 675

# $CHARw. Informat

Reads character data with blanks.

Categories: CAS
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, 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, whereas the $*w*. informat does.
- Use the following table 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

This example uses the TRANSLATE function to replace specific characters in a character expression. The TRANSLATE function converts the spaces in Name2 to # symbols so that the blanks in the value are more obvious.

```
data one;
   name=' X YZ';
   name2=translate(input(name,$char5.),'#',' ');
```

```
    put name ;
    put name2 ;
run;
```

The character # represents a blank space.

SAS writes the following output to the log:

```
X YZ
#X#YZ
```

# $CHARZBw. Informat

Converts binary 0s to blanks.

Categories: CAS
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

This hexadecimal input string, `'58595A0000'x`, is “XYZ” followed by two null characters. The informat converts the null characters to spaces, indicated by hexadecimal ‘20’x in the output.

```
data a;
  x='58595A0000'x;
  y=input(x,$charzb5.);
  put y=;
  put y=$hex10;
run;
```

SAS writes the following output to the log:

```
y=58595A2020
y=XYZ
```

# $EBCDICw. Informat

Converts EBCDIC character data to native format.

Categories: CAS
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:** Anytime a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option in either the

ASCII or EBCDIC environment. 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= option in the FILENAME or INFILE statement, the ENCODING= option is required because these options require certain characters (for example, quotation marks, commas, and blanks) in the session encoding. Encoding-specific informats should be reserved for use with true binary files (that is, files that 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

```
data _null_;
   a=input('qrs',$ebcdic3.);
   b=input('QRS',$ebcdic3.);
   c=input('+:>',$ebcdic3.);
   put a= hex6. b= hex6. c= hex6.;
run;
```

| Variable | Result |
|---|---|
| a | C9CACB |
| b | E9EAEB |
| c | 8B9A9E |

**Note:** The results are hexadecimal representations of codes for characters. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one character value.

# $HEXw. Informat

Converts hexadecimal data to character data.

Categories: CAS
Character

See: "$HEX Informat: UNIX" in *SAS Companion for UNIX Environments*
"$HEXw.: 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, $HEX*w*. reads *w*–1 hexadecimal characters.

**Default** 2

**Range** 1–32767

# Details

The $HEX*w*. informat converts every two digits of hexadecimal data to 1 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 to 1 byte of numeric data.

# Example

```
data test;
   name='6C6C';
   name2=input(name,$hex4.);
run;
```

***Table 4.3*** *Output on EBCDIC and ASCII Systems*

| Variable | EBCDIC Results | ASCII Results |
|---|---|---|
| 6C6C | %% | 11 |

# $N8601Bw.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 notation.

| | |
|---|---|
| Categories: | CAS<br>Date and Time<br>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** 0–6

## Details

The $N8601B informat reads ISO 8601 duration, interval, and datetime values as character data for these 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* | 2012-09-15T15:53:00/2014-11-13T00:00:00 |
| | *yyyymmdd*T*hhmmss.fff*/ *yyyymmdd*T*hhmmss.fff* | 20120915T155300/20141115T120000 |
| | 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.

**Note:** Using a month value that is greater than 12 with a datetime value causes an error. For example, the value 20121415T000000/2014-09-15T00:00:00 causes an error because the value for the month is 14.

# 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

The resulting value of Date2 can be formatted so that it is displayed like Date with the $N8601E format.

```
data one;
date='2018-09-15T15:53:00';
dt2=input(date,$n8601b20.);

put dt2 /
    dt2=$N8601E.;
run;
```

SAS writes the following output to the log:

```
2018915155300FFD
dt2=2018-09-15T15:53:00
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# $N8601Ew.d Informat

Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.

| | |
|---|---|
| Categories: | CAS<br>Date and Time<br>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 these 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*n*s.*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

The resulting value of Date2 can be formatted so that it is displayed like Date with the $N8601E format.

```
data one;
   date='2018-09-15T15:53:00';
   date2=input(date,$n8601e20.);

   put date2  /
```

```
        date2=$N8601E.;
run;
```

SAS writes the following output to the log:

```
2012915155300FFD
date2=2018-09-15T15:53:00
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# $OCTALw. Informat

Converts octal data to character data.

Categories: CAS
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*. reads.

**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 the octal notation 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 it to the numeric equivalents.

## Example

```
data _null_;
   a=input('141162164040040', $octal15.);
   b=input('162151143145040', $octal15.);
   c=input('142141156153040', $octal15.);
   put a= b= c=;
run;
```

SAS writes the following output to the log:

```
a=art b=rice c=bank
```

# $PHEXw. Informat

Converts packed hexadecimal data to character data.

Categories: CAS
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 is 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 represents. 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 it to numeric data.

## Example

```
data _null_;
   a=input('0001111000001111',$phex.);
   put a=;
run;
```

SAS writes the following to the log:

```
a=303
```

# $QUOTEw. Informat

Removes matching quotation marks from character data.

Categories: CAS
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

```
data one;
   name='SAS';
   name2=input(name,$quote7.);
   put name2= ;
run;
```

SAS writes the following output to the log:

```
name2=SAS
```

# $UPCASEw. Informat

Converts character data to uppercase.

Categories: CAS
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

```
data test;
   name='sas';
   name2=input(name,$upcase.);
   put name2 ;
run;
```

| Variable | Result |
|---|---|
| ----+----1 | |
| sas | SAS |

# $UUIDw. Informat

Converts a Universally Unique Identifier (UUID) to binary data.

## Syntax

**$UUIDw.**

## Arguments

***w***

specifies the width of the input field.

**Range** 1–36

## Details

The input for the informat should be a character representation of a UUID. That representation consists of eight hexadecimal characters, a hyphen, four hexadecimal digits, a hyphen, four hexadecimal characters, a hyphen, and twelve hexadecimal characters. A total of thirty-two hexadecimal characters convert to 16 binary bytes. The input typically uses lowercase for the hexadecimal characters a–f,

although the characters can be uppercase. Leading or trailing blanks are acceptable, but there must be thirty-six continuous characters between blanks.

## Example

```
data _null_;
   length x $16;
   x = input('1548611f-b8cb-6a47-a721-a6fd284e74f2',$uuid36.);
   put x $hex.;
run;
```

SAS writes the following output to the log:

```
1548611FB8CB6A47A721A6FD284E74F2
```

# $VARYINGw. Informat

Reads character data of varying length.

Categories: CAS
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

```
data one;
   input fwidth 1. name $varying9. fwidth;
   put name;
   datalines;
5shark
3sunfish
8bluefish
;
run;
```

SAS writes the following to the log.

```
shark
sun
bluefish
```

**Note:** The result of the second variable, `3sunfish`, is truncated.

## Example 2: Obtaining a Current Record Length Indirectly

Use the LENGTH= option in the INFILE statement to obtain a record length indirectly. The input variables 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.

| Variable | Result |
|---|---|
| `----+----1----+----2` | |
| `PATEL  CHEMISTRY` | `PATEL  CHEMISTRY` |
| `JOHNSON  GEOLOGY` | `JOHNSON  GEOLOGY` |
| `WILCOX  ART` | `WILCOX  ART` |

# $w. Informat

Reads standard character data.

Categories: CAS
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, whereas $*w*. does both.

# Example

This example truncates the leading blanks on the `name` statement.

```
data one;
   name='  XYZ';
   name2=input(name,$5.);
put name2;
run;
```

SAS writes the following output to the log:

```
XYZ
```

# ANYDTDTEw. Informat

Reads and extracts the date value from various date, time, and datetime forms.

Categories: CAS
Date and Time

Restriction: Do not use a period (.) as a separator in the date part or between the date and time parts of date-time values. SAS might consider a period to be an invalid separator in the date part of ANYDTDTE values when running in certain environments. The time part of

the date-time value can contain a period. For example, 01JAN2018 14:30:08.5 is a valid date-time value that contains a period in the time part.

# Syntax

**ANYDTDTE***w*.

## Syntax Description

***w***

specifies the width of the input field.

**Default** 9

**Range** 5–60

# Details

The ANYDTDTE informat reads input data that corresponds to any of the following informats or date, time, or datetime forms. Then, the informat extracts the date part from the derived value.

| Informat or Form of Input | Example Data | Informat or Form of Input | Example Data |
|---|---|---|---|
| DATE | 01JAN18<br>01JAN2018 | MONYY | JAN18<br>JAN2018 |
| DATETIME | 01JAN18 14:30:08<br>01JAN2018 14:30:08.5 | TIME | 14:30<br>14:30:08.05 |
| DDMMYY | 010118<br>01012018 | YMDDTTM | 18-01-01 11:23 |
| JULIAN | 18001<br>2018001 | YYMMDD | 180101<br>20180101 |
| MDYAMPM | 01-01-18 3:53 pm | YYQ | 18Q1<br>2018Q1 |
| MMDDYY | 010118<br>01012018 | YY<YY>*x*MM [1] | 18/01<br>2018-01 |
| MM*x*YY<YY> [1] | 01/18 | *month-day-year* | January 1, 2018 |

| Informat or Form of Input | Example Data | Informat or Form of Input | Example Data |
|---|---|---|---|
| | 01-2018 | | |

1 *x* is a special character that separates the month from the year.

If the input value is a time-only value, 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.

Input data that contains colons is interpreted as time data. For example, 2013:12 is interpreted as 2013 hours and 12 minutes, not as the year 2013 and the month 12. The date result of reading a time value is 0.

# 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

```
data test;
   dateinfo='01JAN2018 14:30:08.5';
   sasdate=input(dateinfo,anydtdte21.);
   put sasdate ;
   put sasdate date9.;
run;
```

SAS writes the following output to the log.

```
21185
01JAN2018
```

# See Also

**Informats:**

- “ANYDTDTMw.” on page 589
- “ANYDTTMEw.” on page 592
- “DATEw.” on page 617
- “DATETIMEw.” on page 618
- “DDMMYYw.” on page 621
- “JULIANw.” on page 645

- “MDYAMPMw.d” on page 647
- “MMDDYYw.” on page 649
- “MONYYw.” on page 651
- “TIMEw.” on page 702
- “YMDDTTMw.d” on page 718
- “YYMMDDw.” on page 720
- “YYQw.” on page 724


# ANYDTDTMw. Informat

Reads and extracts datetime values from various date, time, and datetime forms.

Categories: CAS
Date and Time

Restriction: Do not use a period (.) as a separator in the date part or between the date and time parts of date-time values. SAS might consider a period to be an invalid separator in the date part of ANYDTDTM values when running in certain environments. The time part of the date-time value can contain a period. For example, 01JAN2018 14:30:08.5 is a valid date-time value that contains a period in the time part.

Interaction: If an input datetime value contains a special character for formatting characters and the character is not B, C, N, P, or S, the ANYDTDTM*w*. informat reads only the date portion of the input and the time is set to 0. For example, for an EN_US locale, an input value of '150501X1' results in 01MAY15:00:00:00.

## Syntax

**ANYDTDTM***w*.

## Syntax Description

***w***

specifies the width of the input field.

**Default** 19

**Range** 1–60

## Details

The ANYDTDTM informat reads data that is in the form of any of the following informats or datetime forms. Then, the format extracts the datetime part from the derived value:

| Informat or Form of Input | Example Data |
|---|---|
| DATE | 01JAN18<br>01JAN2018 |
| DATETIME | 01JAN18 14:30:08<br>01JAN2018 14:30:08.5 |
| DDMM<YY>YY [2, 3] | 010118<br>01012018 |
| JULIAN | 18001<br>2018001 |
| MMDD<YY>YY [2, 3] | 010118<br>01012018 |
| MM*x*<YY>YY [1, 2] | 01/18<br>01-2018 |
| MDYAMPM [3] | 01/01/18 02:30:08 AM<br>01/01/2018 02:30:08 AM |
| MONYY | JAN18<br>JAN2018 |
| TIME | 14.30<br>14:30:08.05 |
| <YY>YYMMDD [2] | 180101<br>20180101 |
| <YY>YYQ [2] | 18Q1<br>2018Q1 |
| <YY>YY*x*MM [1, 2] | 18/01<br>2018/01 |
| *month-day-year* | January 1, 2018 |
| *month-day-year hh:mm:ss* | January 1, 2018 12:00:00 |

1 *x* is a special character that separates the month from the year.
2 <YY> indicates the century is optional.
3 If 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, SAS assumes a date of 01JAN1960. If the input value is a date-only value, 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, 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 these 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 |
| Multiple periods in the value indicate that the period is a delimiter for dates and the value is not a time value. | 12.25.2018 |

# 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

```
data test;
   dateinfo='01JAN2018 14:30:08.5';
   sasdt=input(dateinfo,anydtdtm21.);
   put sasdt ;
   put sasdt datetime20.;
run;
```

SAS writes the following output to the log:

```
1830436208.5
01JAN2018:14:30:09
```

## See Also

**Informats:**


- "ANYDTDTEw." on page 586
- "ANYDTTMEw." on page 592
- "DATEw." on page 617
- "DATETIMEw." on page 618
- "DDMMYYw." on page 621
- "JULIANw." on page 645
- "MMDDYYw." on page 649
- "MONYYw." on page 651
- "TIMEw." on page 702
- "YYMMDDw." on page 720
- "YYQw." on page 724


# ANYDTTMEw. Informat

Reads and extracts time values from various date, time, and datetime forms.

Categories: CAS
Date and Time

Restriction: Do not use a period (.) as a separator in the date part or between the date and time parts of date-time values. SAS might consider a period to be an invalid separator in the date part of ANYDTTME values when running in certain environments. The time part of the date-time value can contain a period. For example, 01JAN2018 14:30:08.5 is a valid date-time value that contains a period in the time part.

## Syntax

**ANYDTTME***w*.

## Syntax Description

***w***

specifies the width of the input field.

**Default** 8

**Range** 1–60

# Details

The ANYDTTME informat reads input data that corresponds to any of these informats or forms:

| Informat or Form of Input | Example Data | Informat or Form of Input | Example Data |
|---|---|---|---|
| DATE | 01JAN18<br>01JAN2018 | MONYY | JAN18<br>JAN2018 |
| DATETIME | 01JAN18 14:30:08<br>01JAN2018 14:30:08.5 | YYMMDD | 180101<br>20180101 |
| DDMMYY | 010118<br>01012018 | YYQ | 18Q1<br>2018Q1 |
| JULIAN | 18001<br>2018001 | YYQ | 18Q1<br>2018Q1 |
| MMDDYY | 010118<br>01012018 | *month-day-year* | January 1, 2018<br>2018-01 |

If the input value is a time-only value, SAS assumes a date of 01JAN1960. If the input value is a date value only, 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 these 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 |
| Multiple periods in the value indicate that the period is a delimiter for dates and the value is not a time value. | 12.25.2018 |

## 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

```
data test;
   dateinfo='01JAN2018 14:30:08.5';
   sastime=input(dateinfo,anydttme21.);
   put sastime ;
   put sastime time8.2;
run;
```

| Variable | Result |
|---|---|
| put sastime; | 14:30:09 |
| put sastime time8.2; | 01JAN2018:14:30:09 |

## See Also

**Informats:**


- “ANYDTDTEw.” on page 586
- “ANYDTDTMw.” on page 589
- “DATEw.” on page 617
- “DATETIMEw.” on page 618
- “DDMMYYw.” on page 621
- “JULIANw.” on page 645
- “MMDDYYw.” on page 649
- “MONYYw.” on page 651
- “TIMEw.” on page 702
- “YYMMDDw.” on page 720
- “YYQw.” on page 724


# B8601Clw.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

Restriction: This informat is not supported in a DATA step that runs in CAS.

# 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.

***Table 4.4*** *Digits and Represented Centuries*

| Digit | 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 |

| Digit | Century |
|---|---|
| | 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

```
data one;
   date='118100112153025';
   date2=input(date,B8601CI.);
   put date2 /
       date2 datetime25.2;
run;
```

SAS writes the following output to the log:

```
1854015330.3
01OCT2018:12:15:30.25
```

# B8601DAw. Informat

Reads date values that are specified using the ISO 8601 base notation *yyyymmdd*.

| | |
|---|---|
| Categories: | Date<br>ISO 8601 |
| Alignment: | Left |
| Alias: | ND8601DA*w* |
| Restrictions: | This informat is not supported in a DATA step that runs in CAS. |

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

# 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.

# Example

```
data _null_;
   input d b8601da.;
   put d;
   datalines;
20180915
2018
;
run;
```

SAS writes the following to the log.

```
21442
21185
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# B8601DJw.d Informat

Reads a Java date and time value that is in the form *yyyymmddhhmmss<ffffff>*.

Categories: Date and Time
ISO 8601

Alignment: Left

Restriction: This informat is not supported in a DATA step that runs in CAS.

## 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*w.d* 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*w.d* 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 September 1, 2018 at 7:30:00.33 a.m. is written as 2018090107300033.

ISO 8601 date and time values include a T. For example, the date September 1, 2018 at 7:30:00.33 a.m. is written as 20180901T07300033.

## Example

```
data one;
   date='2018090107300033';
   date2=input(date,B8601DJ.);
   put datetm2 /
       datetm2 datetime25.2;
run;
```

SAS writes the following output to the log:

```
1851406200.3
01SEP2018:07:30:00.33
```

# B8601DNw. 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
Date and Time
ISO 8601

Alignment: Left

Alias: ND8601DN*w*

Restrictions: This informat is not supported in a DATA step that runs in CAS.
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

```
data one;
   date='20180915';
   datetm=input(date,B8601DN.);
   put datetm /
       datetm datetime20.;
run;
```

SAS writes the following output to the log:

```
1852588800
15SEP2018:00:00:00
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# B8601DTw.d Informat

Reads datetime values that are specified using the ISO 8601 basic notation *yyyymmdd*T*hhmmss<ffffff>*.

| | |
|---|---|
| Categories: | Date and Time<br>ISO 8601 |
| Alignment: | Left |
| Aliases: | B8601LX*w.d*<br>ND8601DT*w.d* |
| Restrictions: | This informat is not supported in a DATA step that runs in CAS.<br>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** 14–26

***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

```
data _null_;
   input d b8601dt.;
   put d;
   datalines;
20180915T155300
2018
;
run;
```

SAS writes the following to the log.

```
1852645980
1830384000
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# B8601DXw. Informat

Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, reads the local date and time by using the ISO 8601 datetime and time zone basic notation *yyyymmdd*T*hhmmss* +|-*hhmm*

Restriction: This informat is not supported in a DATA step that runs in CAS.

See: The B8601DX*w.* informat uses the B8601DZ*w.d* informat to read data. See "B8601DZw.d" on page 603.

## Syntax

**B8601DX*w.***

# B8601DZw.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

Aliases: B8601DX*w.d*
ND8601DZ*w.d*

Restrictions: This informat is not supported in a DATA step that runs in CAS.
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** 15–35

***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 these 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.

# Example

```
data one;
   input d b8601dz.;
   put d;
   put d datetime.;
   datalines;
20180915T155300+0500
;
run;
```

SAS writes the following output to the log:

```
1852627980
15SEP18:10:53:00
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# B8601LXw. Informat

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

Restriction: This informat is not supported in a DATA step that runs in CAS.

See: The B8601LX*w.d* informat uses the B8601DT*w.d* informat to read data. See "B8601DTw.d" on page 601.

## Syntax

**B8601LX*w.d***

# B8601TMw.d Informat

Reads time values that are specified using the ISO 8601 basic notation *hhmmss<ffffff>*.

Categories: Time
ISO 8601

Alignment: Left

Alias: ND8601TM*w.d*

Restrictions: This informat is not supported in a DATA step that runs in CAS.
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

```
data one;
   input d b8601tm.;
   put d;
   datalines;
155300
run;
```

SAS writes the following output to the log:

```
57180
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# B8601TXw. Informat

Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, reads the local time by using the ISO 8601 basic time notation *hhmmss*+|–*hhmm*.

Restriction: This informat is not supported in a DATA step that runs in CAS.

See: The B8601TX*w.* informat uses the B8601TZ*w.d* informat to read data. See "B8601TZw.d" on page 607.

## Syntax

**B8601TX*w.***

# B8601TZw.d Informat

Reads time values that are specified using the ISO 8601 basic time notation *hhmmss<ffff>*+|–*hhmm* or *hhmmss<ffffff>*Z.

Categories: Time
ISO 8601

Alignment: Left

Aliases: B8601TX*w.d*
ND8601TZ*w.d*

Restrictions: This informat is not supported in a DATA step that runs in CAS.
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** 7–20

***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 these 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

```
data _null_;
   input d b8601tz.;
   put d= d=time8.;;
   datalines;
202401-0500
202401Z
202401+0000
run;
```

SAS writes the following output to the log:

```
d=5041 d=1:24:01
d=73441 d=20:24:01
d=73441 d=20:24:01
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# BINARYw.d Informat

Converts positive binary values to integers.

Categories: CAS
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 contains 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

```
data one;
   input @1 value binary8.1;
   put value;
   datalines;
00001111
run;
```

SAS writes the following output to the log:

```
1.5
```

# BITSw.d Informat

Extracts bits.

Category: Numeric

Restriction: This informat is not supported in a DATA step that runs in CAS.

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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 with many pieces of information packed into single bytes.

## Example

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

```
data _null_;
   a=input('B',bits4.1);
   put a=;
run;
```

SAS writes the following output to the log:

```
a=8
```

# BZw.d Informat

Converts blanks to 0s.

Category: Numeric

Restriction: This informat is not supported in a DATA step that runs in CAS.

## 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 contains 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 variable 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

```
data _null_;
   input @1 x bz4.;
   put x;
   datalines;
34
-2
-2 1
.
run;
```

SAS writes the following output to the log:

```
3400
-200
-201
0
```

# CBw.d Informat

Reads standard numeric values from column-binary files.

Categories: CAS
Column Binary

## Syntax

**CB***w.d*

## Syntax Description

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

**Default** 1

**Range** 1–32

***d***
0–31, specifies the power of 10 by which to divide the value. SAS uses the *d* value even if the data contains 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 2 bytes and ignores the two high-order bits of each byte. If the punch codes are valid, SAS stores the equivalent numeric value in the variable that you specify. If the combinations are not valid, SAS assigns the variable a missing value and sets the automatic variable _ERROR_ to 1.

## Example: Examples

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

| Variable[1] | Result |
|---|---|
| ----+----1 | |
| 0009 | 9 |

1 The variable 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

- “Description of Column-Binary Data Storage” in *SAS Programmer’s Guide: Essentials*

**Informats**

- “$CBw.” on page 566
- “PUNCH.d” on page 669
- “ROWw.d” on page 675

# COMMAw.d Informat

Removes embedded characters.

Categories: CAS
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 contains 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

```
data test;
   value='$1,000,000';
   value2=input(value,comma10.);
   put value2 ;
run;
```

SAS writes the following output to the log:

```
1000000
```

# COMMAXw.d Informat

Removes embedded periods, blanks, dollar signs, percent signs, hyphens, and close parentheses 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.

Categories: CAS
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 contains 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 closing 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

```
data one;
   input @1 x commax10.;
   put x;
   datalines;
$1.000.000
1.234,56
(500)
run;
```

SAS writes the following output to the log:

```
1000000
1234.56
-500
```

# DATEw. Informat

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

Categories: CAS
Date

## 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 four-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*
are 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 special characters. Ensure that the width of the input field allows space for blanks and special characters.

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

## Example

```
data test;
   date='28JAN2020';
   sasdate=input(date,date9.);
   put sasdate;
   put sasdate date9;
run;
```

| Variable | Result |
|---|---|
| ----+----1----+----20----+ | |
| put sasdate ; | 21942 |
| put sasdate date9; | 28JAN2020 |

## See Also

**Formats:**

- "DATEw." on page 169

**Functions:**

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

**System Options:**

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

# DATETIMEw. Informat

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

Categories: CAS
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*
are 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 that range 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 part of 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* 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 YMDDTTM*w.d* informat reads datetime values with required separators in the form *<yy>yy-mm-dd/hh:mm:ss.ss*.

# Example

```
data test;
   date='04MAY2018:13:33:18';
   sasdate=input(date,datetime.);
   put sasdate ;
   put sasdate datetime20.;
run;
```

| Variable | Result |
|---|---|
| | ----+----1----+----2 |
| put sasdata ; | 1841059998 |
| put sasdata datetime20.; | 04MAY2018:13:33:18 |

# See Also

- “Definitions for Dates, Times, and Intervals” on page 9

**Formats:**

- “DATEw.” on page 169
- “DATETIMEw.d” on page 172
- “TIMEw.d” on page 465

**Functions:**

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

**Informats:**

- “DATEw.” on page 617
- “MDYAMPMw.d” on page 647
- “TIMEw.” on page 702
- “YMDDTTMw.d” on page 718

**System Options:**

- “YEARCUTOFF=” in *SAS System Options: Reference*

# DDMMYYw. 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 two or four digits.

Categories: CAS
Date

## 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 separator that can be any special character or a blank space.

If you use separators, place them between all the values. Blanks can also be placed before and after the date. Ensure that the width of the input field allows space for blanks and special characters.

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

## Example

```
data test;
   date='04/05/2018';
   sasdate=input(date,ddmmyy10.);
   put sasdate ;
   put sasdate date9.;
run;
```

| Variable | Result |
|---|---|
| | ----+----1----+ |
| put sasdate ; | 21308 |
| put sasdate date9.; | 04MAY2018 |

## See Also

**Formats:**

- “DATEw.” on page 169
- “DDMMYYw.” on page 175
- “MMDDYYw.” on page 235
- “YYMMDDw.” on page 498

**Functions:**

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

**Informats:**

- “DATEw.” on page 617
- “MMDDYYw.” on page 649
- “YYMMDDw.” on page 720

**System Options:**

- “YEARCUTOFF=” in *SAS System Options: Reference*

# E8601DAw. Informat

Reads date values that are specified using the ISO 8601 extended notation *yyyy-mm-dd*.

Categories: CAS
Date

ISO 8601

Alignment: Left

Alias: IS8601DA*w*

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

```
data one;
   input eda e8601da.;
   put eda;
   datalines;
2018-09-15
;
run;
```

SAS writes the following output to the log:

```
21442
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# E8601DNw. 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: | CAS<br>Date<br>ISO 8601 |
| Alignment: | Left |
| Alias: | IS8601DN*w* |
| 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

```
data one;
   input edn e8601dn.;
   put edn;
   datalines;
2018-09-15
;
run;
```

SAS writes the following output to the log:

```
1852588800
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# E8601DTw.d Informat

Reads datetime values that are specified using the ISO 8601 extended notation *yyyy-mm-dd*T*hh:mm:ss.<ffffff>*.

| | |
|---|---|
| Categories: | CAS<br>Date and Time<br>ISO 8601 |
| Alignment: | Left |
| Aliases: | E8601LX*w.d*<br>IS8601DT*w.d* |
| 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** 16–26

***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.

**Note:** If you specify a width of 16, SAS assumes that the value for seconds is 0 and omits them from the output.

# Example

```
data one;
   input edt e8601dt.;
   put edt;
   datalines;
2018-09-15T15:53:00
;
run;
```

SAS writes the following output to the log:

```
1852645980
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# E8601DXw. Informat

Adjusts a Coordinated Universal Time (UTC) datetime value to the user's local date and time. Then, reads the local date and time by using the ISO 8601 datetime and time zone extended notation *yyyy-mm-dd*T*hh:mm:ss*+|–*hh:mm*.

See: The E8601DX*w.* informat uses the E8601DZ*w.d* informat to read data. See "E8601DZw.d" on page 627.

## Syntax

**E8601DX*w.***

# E8601DZw.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: CAS
Date and Time
ISO 8601

Alignment: Left

Aliases: E8601DX*w.d*
IS8601DZ*w.d*

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 these 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

```
data one;
   input edz e8601dz.;
   put edz= edz=datetime22
   datalines;
2018-09-15T15:53:00Z
;
run;

data one;
   input edz e8601dz28.2;
   put edz= edz=datetime22
   datalines;
2018-09-15T15:53:00+03:00
;
run;
```

SAS writes the following output to the log:

```
edz=1852645980 edz=1852645980
edz=1852635180 edz=1852635180
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# E8601LXw. Informat

Reads datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation *yyyy-mm-dd*T*hh:mm:ss*+|–*hh:mm*.

Restriction: This informat is not supported in a DATA step that runs in CAS.

See: The E8601LX*w.* informat uses the E8601DT*w.d* informat to read data. See "E8601DTw.d" on page 625.

## Syntax

**E8601LX*w.***

# E8601LZw.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: | Time<br>ISO 8601 |
| Alignment: | Left |
| Alias: | IS8601LZ*w.d* |
| Restriction: | This informat is not supported in a DATA step that runs in CAS. |
| 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 more data follows the time 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 these 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

```
data one;
   input edz e8601lz.;
   put etz= etz=time8.;
   datalines;
09:13:21+02:00
23:43:44Z
;
run;
```

SAS writes the following output to the log:

```
edz=8001 edz=2:13:21
edz=85424 edz=23:43:44
```

# See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# E8601TMw.d Informat

Reads time values that are specified using the ISO 8601 extended notation *hh:mm:ss.<ffffff>*.

| | |
|---|---|
| Categories: | CAS<br>Time<br>ISO 8601 |
| Alignment: | Left |
| Alias: | IS8601TM*w.d* |
| 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

```
data one;
   input edm e8601tm.;
   put edm;
   datalines;
15:53:00
;
run;
```

SAS writes the following output to the log:

```
57180
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# E8601TXw. Informat

Adjusts a Coordinated Universal Time (UTC) value to the user's local time. Then, reads the local time by using the ISO 8601 extended time notation *hh:mm:ss*+|–*hh:mm*.

Restriction: This informat is not supported in a DATA step that runs in CAS.

See: The E8601TX*w.* informat uses the E8601TZ*w.d* informat to read data. See "E8601DZw.d" on page 627.

## Syntax

**E8601TX*w.***

# E8601TZw.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: CAS
Time
ISO 8601

Alignment: Left

Aliases: E8601TX*w.d*
IS8601TZ*w.d*

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 more data follows the time 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 these ISO 8601 extended notations:

- *hh:mm:ss+|–hh:mm.<ffffff>*
- *hh:mm:ssZ*

*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

```
data one;
   input etz e8601tz.;
   put etz= etz=time8.;
   datalines;
23:43:44-05:00
23:43:44Z
;
run;
```

SAS writes the following output to the log:

```
etz=17024 etz=4:43:44
etz=85424 etz=23:43:44
```

## See Also

"Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations" on page 540

# FLOATw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

# 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 4-byte floating-point number is the same as a truncated 8-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 4-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. |
| Fortran | REAL*4 |
| C | float |
| IBM 370 ASM | E |

| Language | Float Notation |
| --- | --- |
| PL/I | FLOAT BIN(21) |

## Example

```
input x float4.;
```

| Variable[1] | Result |
| --- | --- |
| ----+----1----+----2 | |
| 3F800000 | 1 |

1 The variable is a hexadecimal representation of a binary number that is stored in IEEE form.

# HEXw. Informat

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

Categories: CAS
Numeric

See: "HEX Informat: UNIX" in *SAS Companion for UNIX Environments*
"HEXw.: 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

```
data _null_;
   a=input('88F'x,hex3.);
   b=input('4152000000000000'x,hex16.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=2191 b=4718592
```

# HHMMSSw. Informat

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

Categories: CAS
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 these 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, SAS reads the data from left to right as hours, minutes, and seconds. For data that is less than six digits, SAS follows these rules:

- If the number has an odd number of digits, SAS pads a zero to the left of the first digit. Then, SAS pads after the last digit to the right with zeros until there are six digits.
- If the number has an even number of digits, SAS pads zeros to the right of the last digit until there are six digits.

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 010000 or 1:00:00.

02 is the same as 020000 or 2:00:00.

124 is the same as 0124000 or 1:24:00.

1435 is the same as 143500 or 14:35:00.

20345 is the same as 020345 or 2:03:45.

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. SAS reads the third and forth digits from the right as minutes. SAS reads the remaining digits to the left of the minutes 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), SAS reads the two digits before the colon as hours. SAS reads the two digits after the colon as minutes. The number of seconds is 0.

If a colon is omitted between minutes and seconds, as in 12:3400, SAS reads the 3400 as 3400 minutes. 3400 minutes adds 56 hours and 40 minutes to the 12 hours, resulting in 68:40:00.

## Example

```
data test;
   time='12:45:44';
   sastime=input(time,hhmmss.);
   put sastime ;
   put sastime time8.;
run;
```

| Variable | Result |
|---|---|
| | `----+----1----+` |
| `put sastime ;` | `45944` |
| `put sastime time8.;` | `12:45:44` |

## See Also

**Informats:**

- "TIMEw." on page 702

# IBw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

See: "IB Informat: UNIX" in *SAS Companion for UNIX Environments*
"IBw.d: 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 532 .

# 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 "Reading Data Generated on Big Endian and Little Endian Platforms" on page 533.

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

# 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.

```
data one;
```

```
   result=input('0080'x,ib4.);
   put result;
run;
```

| SAS Statement | Result on Big Endian Platforms | Result on Little Endian Platforms |
|---|---|---|
| put result; | 128 | -32768 |

## See Also

**Informats:**

- "IBRw.d" on page 642

# IBRw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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 532 .

# 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.

In 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 "Reading Data Generated on Big Endian and Little Endian Platforms" on page 533.

To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

# Example

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

```
data one;
   x=input('0100'x,ibr2.);
   y=input('0001'x,ibr2.);
   result=input('80000000'x,ibr4.);
   put x;
   put y;
   put result;
run;
```

| SAS Statement | Result on Big Endian Platforms | Result on Little Endian Platforms |
|---|---|---|
| `put x;` | 1 | 1 |
| `put y;` | 256 | 256 |
| `put result;` | 128 | 128 |

## See Also

**Informats:**

- "IBw.d" on page 640

# IEEEw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## Syntax

**IEEE***w.d*

## Syntax Description

***w***

specifies the width of the input field.

**Default** 4

**Range** 3–8

**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 1 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 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). Truncation is performed by programs 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* informat allows other lengths, which enables you to read data from files that contain space-saving truncated data.

## Example

The variables are hexadecimal representations of binary numbers that are stored in IEEE format.

```
data _null_;
   a=input('3F800000'x,ieee4.);
   b=input('3FF0000000'x,ieee5.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=1 b=1
```

# JULIANw. Informat

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

Categories: CAS
Date and Time

See: “Julian Date Formats and Astronomical Dates” on page 10

## 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 these 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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

# Example

```
data test;
   date='2018099';
   sasdate=input(date,julian7.);
   put sasdate ;
   put sasdate date9.;
run;
```

| Variable | Result |
|---|---|
| | `----+----1----+` |
| `put sasdate ;` | `21283` |
| `put sasdate date9.;` | `09APR2018` |

## See Also

**Formats:**

- "JULIANw." on page 232

**Functions:**

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

**System Options:**

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

# MDYAMPMw.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 two or four digits.

| | |
|---|---|
| Categories: | CAS<br>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

```
data test;
   date='04-30-18 3:27 pm';
   sasdate=input(date,mdyampm.);
   put sasdate ;
   put sasdate datetime20.;
run;
```

| Variable | Result |
|---|---|
| `put sasdate ;` | `1840721220` |
| `put sasdate datetime20.;` | `30APR2018:15:27:00` |

## See Also

**Informats:**

- "DATETIMEw." on page 618
- "YMDDTTMw.d" on page 718

# MMDDYYw. Informat

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

Categories: CAS
Date

## 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 these 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 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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

# Example

```
data test;
   date='05/04/2018';
   sasdate=input(date,mmddyy10.);
   put sasdate ;
   put sasdate date9.;
run;
```

| Variable | Result |
|---|---|
| ----+----1----+ | |
| put sasdate ; | 21308 |
| put sasdate date9.; | 04MAY2018 |

# See Also

**Formats:**


- “DATEw.” on page 169
- “DDMMYYw.” on page 175
- “MMDDYYw.” on page 235

- "YYMMDDw." on page 498

**Functions:**

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

**Informats:**

- "DATEw." on page 617
- "DDMMYYw." on page 621
- "YYMMDDw." on page 720

**System Options:**

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

# MONYYw. Informat

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

Categories: CAS
Date

## 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 these forms:

- *mmmyy*
- *mmmyyyy*

*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.

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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

# Example

```
data test;
   date='MAY2018';
   sasdate=input(date,monyy7.);
   put sasdate ;
   put sasdate date9.;
run;
```

| Variable | Result |
|---|---|
| ----+----1 | |
| put sasdate ; | 21305 |
| put sasdate date9.; | 01MAY2018 |

# See Also

**Formats:**

- “DDMMYYw.” on page 175
- “MMDDYYw.” on page 235
- “MONYYw.” on page 247
- “YYMMDDw.” on page 498

**Functions:**

- “MONTH” in *SAS Functions and CALL Routines: Reference*
- “YEAR” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “DDMMYYw.” on page 621
- “MMDDYYw.” on page 649

- "YYMMDDw." on page 720

**System Options:**

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

# MSECw. Informat

Reads TIME MIC values.

Categories: CAS
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 returns an 8-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

The input time used in this example is the hexadecimal timestamp ‘0000EA044E65A000’ that corresponds to 5:26:58 p.m.

```
data _null_;
  x=input('0000EA044E65A000'x,msec8.);
  put x=timeampm.;
run;
```

SAS writes the following to the log:

```
x=5:26:58 PM
```

## See Also

**Informats:**

- “TODSTAMPw.” on page 704

# NUMXw.d Informat

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

Categories: CAS
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 contains 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

```
data one;
   input @1 x numx10.;
   put x;
   datalines;
896,48
3064,1
6489
;
run;
```

SAS writes the following output to the log:

```
896.48
3064.1
6489
```

## See Also

**Formats:**

- “NUMXw.d” on page 419
- “w.d” on page 476

# OCTALw.d Informat

Converts positive octal values to integers.

Category: Numeric

Restriction: This informat is not supported in a DATA step that runs in CAS.

# 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

```
data one;
   input @1 x octal13.1;
   put x;
   datalines;
177
;
run;
```

SAS writes the following output to the log:

```
12.7
```

# PDw.d Informat

Reads data that is stored in IBM packed decimal format.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

See: "PD Informat: UNIX" in *SAS Companion for UNIX Environments*
"PDw.d: 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 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 –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: Reading Packed Decimal Data

```
data _null_;
  a=input('000123'x, pd3.);
  put a=;
run;
```

SAS writes the following output to the log:

```
a=123
```

# PDJULGw. Informat

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

Category: Date

Restriction: This informat is not supported in a DATA step that runs in CAS.

## Syntax

**PDJULG***w*.

## Syntax Description

***w***

specifies the width of the input field.

**Default** 4

**Range** 3–16

# Details

The PDJULG*w*. informat reads IBM packed Julian date values in the form *yyyyddd*F:

*yyyy*
is the 2-byte representation of the four-digit Gregorian year.

*ddd*
is the 1 1/2-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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

# Example

```
data a;
x=input('2018003F'x,pdjulg4.);
format x mmddyy10.;
put x=;
run;
```

SAS writes the following output to the log:

```
01/03/2018
```

# See Also

**Formats:**

- “JULDAYw.” on page 230
- “JULIANw.” on page 232
- “PDJULGw.” on page 426
- “PDJULIw.” on page 427

**Functions:**

- “DATEJUL” in *SAS Functions and CALL Routines: Reference*
- “JULDATE” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “JULIANw.” on page 645
- “PDJULIw.” on page 660

**System Options:**

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

# PDJULIw. Informat

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

Category: Date

Restriction: This informat is not supported in a DATA step that runs in CAS.

## Syntax

**PDJULI***w*.

### Syntax Description

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

**Default** 4

**Range** 3–16

## Details

The PDJULI*w*. informat reads IBM packed Julian date values in the form *ccyyddd*F:

*cc*
is the 1-byte representation of a two-digit integer that represents the century.

*yy*
is the 1-byte representation of a two-digit integer that represents the year. The PDJULI*w* informat makes an adjustment to the 1-byte century representation by adding 1900 to the 2-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 1 1/2- 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.

## Example

```
data a;
x=input('0118019F'x,pdjuli4.);
format x mmddyy10.;
put x;
run;
```

SAS writes the following output to the log:

```
01/19/2018
```

## See Also

**Formats:**

- “JULDAYw.” on page 230
- “JULIANw.” on page 232
- “PDJULGw.” on page 426
- “PDJULIw.” on page 427

**Functions:**

- “DATEJUL” in *SAS Functions and CALL Routines: Reference*
- “JULDATE” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “JULIANw.” on page 645
- “PDJULGw.” on page 658

**System Options:**

- “YEARCUTOFF=” in *SAS System Options: Reference*

# PDTIMEw. Informat

Reads packed decimal time of SMF and RMF records.

Categories: CAS
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 SMF and RMF records contain 4 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 1 byte that represents two digits that correspond to hours.

*mm*
is 1 byte that represents two digits that correspond to minutes.

*ss*
is 1 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 in any operating environment.

# Example

The variable 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.

```
data _null_;
   a=input('0142225F'x,pdtime.);
   put a=;
run;
```

SAS writes the following to the log:

```
a=51745
```

# PERCENTw.d Informat

Reads percentages as numeric values.

Categories: CAS
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 contains 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

```
data test;
   value='23.48%';
   percent=input(value,percent7.2);
   put percent;
run;
```

SAS writes the following output to the log:

```
0.2348
```

# PIBw.d Informat

Reads positive integer binary (fixed-point) values.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

See: "PIB Informat: UNIX" in *SAS Companion for UNIX Environments*
"PIBw.d: 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 532 .

# 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 contains 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 "Reading Data Generated on Big Endian and Little Endian Platforms" on page 533.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

# 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.

```
data _null_;
   result=input('A',pib1.);
   put result;
run;
```

```
data _null_;
   x=input('0100'x,pib2.);
   y=input('0001'x,pib2.);
   put x ;
   put y;
run;
```

| SAS Statement | Result on Big Endian Platforms | Result on Little Endian Platforms |
|---|---|---|
| put result; | 193 | 65 |
| put x; | 256 | 1 |
| put y; | 1 | 256 |

## See Also

**Informats:**

- "PIBRw.d" on page 666

# PIBRw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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 and DEC environments in 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 532 .

# 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 contains 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 “Reading Data Generated on Big Endian and Little Endian Platforms” on page 533.

  To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 38.

# Example

```
data _null_;
   x=input('0100'x,pibr2.);
   y=input('0001'x,pibr2.);
   put x  y;
run;
```

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.

| 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" on page 664

# PKw.d Informat

Reads unsigned packed decimal data.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## Syntax

**PK***w.d*

### Syntax Description

***w***

specifies the number of bytes of unsigned packed decimal data. Each byte 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

The variable in this example is a hexadecimal representation of a binary number stored in unsigned packed decimal form.

```
data _null_;
   a='001234'x;
   b=input(a, pk3.);
   put b=;
run;
```

SAS writes the following to the log:

```
b=1234
```

# PUNCH.d Informat

Reads whether a row of column-binary data is punched.

Categories: CAS
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 one;
   input x punch.12;
   put x;
   datalines;
12-7-8
;
run;
```

SAS writes the following output to the log:

```
1
```

## See Also

- "Description of Column-Binary Data Storage" in *SAS Programmer's Guide: Essentials*

**Informats:**

- "$CBw." on page 566
- "CBw.d" on page 613
- "ROWw.d" on page 675

# RBw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

See: "RB Informat: UNIX" in *SAS Companion for UNIX Environments*
"RBw.d: 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 RB*w.d* informat to read real binary information on equipment that conforms to the IEEE standard for floating-point numbers results in a truncated 8-byte number (double-precision) instead of a true 4-byte floating-point number (single-precision).**

## Example

This example uses a hexadecimal representation of a real binary (floating-point) number.

```
data _null_;
   a='0000000000006040'x;
   b=input(a, rb8.);
   put b=;
run;
```

SAS writes the following output to the log:

```
b=128
```

## See Also

**Informats:**

- “IEEEw.d” on page 644

# RMFDURw. Informat

Reads duration intervals of RMF records.

Categories: CAS
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 4 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. Then, the RMFDUR*w*. informat converts the measurement intervals to SAS time values.

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

*mm*
is the 1-byte representation of two digits that correspond to minutes.

*ss*
is the 1-byte representation of two digits that correspond to seconds.

*ttt*
is the 1 1/2-byte representation of three digits that correspond to thousandths of a second.

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

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

# Comparisons

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

# Example

The variable 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 that corresponds to 00:35:52.226.

```
data _null_;
   a=input('3552226F'x,rmfdur4.);
   put a=;
run;
```

SAS writes the following to the log:

```
a=2152.226
```

## See Also

**Informats:**


- "RMFSTAMPw." on page 674
- "SMFSTAMPw." on page 699


# RMFSTAMPw. Informat

Reads time and date fields of RMF records.

Categories: CAS
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 8 bytes of information: 4 bytes of time data that are followed by 4 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. Then, the RMFSTAMP*w*. informat 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 1-byte representation of two digits that correspond to the hour of the day.

*mm*
is the 1-byte representation of two digits that correspond to minutes.

*ss*
is the 1-byte representation of two digits that correspond to seconds.

*cc*
is the 1-byte representation of two digits that correspond to the century.

*yy*
is the 1-byte representation of two digits that correspond to the year.

*ddd*
is the 1 1/2- 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, in 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

The variable 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, 2018, 2:22.25 PM.

```
data one;
   input x $hex16.;
   y=input(x,rmfstamp8.);
   put y ;
   put y datetime16.;
   datalines;
0142225F2618200F
;
run;
```

SAS writes the following output to the log:

```
80739901345
18JUL18:14:22:25
```

# ROWw.d Informat

Reads a column-binary field down a card column.

Category: Column Binary

Restriction: This informat is not supported in a DATA step that runs in CAS.

# 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** 0

**Range** 0–25

# 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, ROW*w.d* assigns the variable a missing value and sets the automatic variable _ERROR_ to 1. If the field has no punches, ROW*w.d* assigns the variable a missing value.

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

# Example

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

| Variable[1] | Result |
|---|---|
| ----+----1 | |
| 00 | |

| Variable[1] | Result |
|---|---|
| 04 | 3 |
| | 1 |
| | . |
| | 5 |

1 The variable 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

- “Description of Column-Binary Data Storage” in *SAS Programmer’s Guide: Essentials*

**Informats:**

- “$CBw.” on page 566
- “CBw.d” on page 613
- “PUNCH.d” on page 669

# S370FFw.d Informat

Reads EBCDIC numeric data.

Categories: CAS
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 is 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 1 byte per digit. Use S370FF*w.d* in 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

The variables are hexadecimal representations of codes for characters. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one character value.

```
data _null_;
   a=input('F1F2F3'x,s370ff3.);
   b=input('F2F4F0'x,s370ff3.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=123 b=240
```

# S370FIBw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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 is 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 532 .

## 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 "Reading Data Generated on Big Endian and Little Endian Platforms" on page 533.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38..

## 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 using a hexadecimal literal.

```
data _null_;
   a=input('0080'x,s370fib2.);
   put a=;
run;
```

SAS writes the following to the log:

```
a=128
```

## See Also

**Informats**

- "S370FIBUw.d" on page 680
- "S370FPIBw.d" on page 685

# S370FIBUw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

# 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 contains 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 is 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 532 .

# 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 "Reading Data Generated on Big Endian and Little Endian Platforms" on page 533.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 38..

## 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 using a hexadecimal literal.

```
data _null_;
   result1=input('7F'x,s370fibu1.);
   result2=input('F6'x,s370fibu1.);
   put result1 +1 result2;
run;
```

| SAS Statement | Result |
|---|---|
| `put result1;` | `result1=127` |
| `put result2;` | `result2=246` |

## See Also

**Informats:**

- “S370FIBw.d” on page 679
- “S370FPIBw.d” on page 685

# S370FPDw.d Informat

Reads packed data in IBM mainframe format.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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 contains 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 in other operating environments.

# Comparisons

- If you use SAS on an IBM mainframe, the S370FPD*w*.*d* and 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 |

## Example

The variable in this example is a hexadecimal representation of a binary number that is stored in packed decimal form. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the input field.

```
data _null_;
   a=input('12345F'x,s370fpd3.);
   put a=;
run;
```

SAS writes the following to the log:

```
a=12345
```

# S370FPDUw.d Informat

Reads unsigned packed decimal data in IBM mainframe format.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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 contains 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* in 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 *n* is the number of digits.

## Example

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

| Variable[1] | Result |
|---|---|
| `----+----1` | |
| `12345F` | `12345` |

1 The variable is a hexadecimal representation of a binary number that is stored in packed decimal form. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the input field.

# S370FPIBw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

# 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 is 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 532 .

# 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 "Reading Data Generated on Big Endian and Little Endian Platforms" on page 533.

  To view a table that compares integer binary notation in several programming languages, see "Integer Binary Notation and Different Programming Languages" on page 38.

## 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.

```
data _null_;
   result=input('0100',s370fpib2.);
   put result;
run;
```

| SAS Statement | Result |
|---|---|
| `put result;` | `12337` |

## See Also

**Informats:**

- "S370FIBw.d" on page 679
- "S370FIBUw.d" on page 680

# S370FRBw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## Syntax

**S370FRB***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

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 in other operating environments.

# Comparisons

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

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

# See Also

**Informats:**

- “RBw.d” on page 670

# S370FZDw.d Informat

Reads zoned decimal data in IBM mainframe format.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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 contains decimal points, the *d* value is ignored. This argument is optional.

**Default** 0

**Range** 0–31

## Details

Zoned decimal data is similar to standard decimal data in that every digit requires 1 byte. However, the value's sign is stored in the last byte, along with the last digit.

Use S370FZD*w.d* in 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

The variable contains a hexadecimal representation of a binary number stored in zoned decimal format in an IBM mainframe operating environment. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the input field.

```
data _null_;
   a=input('F1F2C3'x,s370fzd3.);
   b=input('F1F2D3'x,s370fzd3.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=123 b=-123
```

## See Also

**Informats:**

- “ZDw.d” on page 725

# S370FZDBw.d Informat

Reads zoned decimal data in which 0s have been left blank.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

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 in other operating environments to read zoned decimal data from IBM mainframe files.

# Example

The variables contain a hexadecimal representation of a binary number that is stored in zoned decimal format in an IBM mainframe operating environment. Two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the input field.

```
data _null_;
   a=input('40404040F14040C0'x,s370fzdb8.);
   b=input('4040404040F1F2D3'x,s370fzdb8.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=1000 b=-123
```

# S370FZDLw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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* in 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 *n* is the number of digits.

## Example

The variables contain a hexadecimal representation of a binary number stored in zoned decimal format in an IBM mainframe operating environment. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the input field.

```
data _null_;
   a=input('C1F2F3'x,s370fzdl3.);
   b=input('D1F2F3'x,s370fzdl3.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=123 b=-123
```

# S370FZDSw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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* in 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 *n* is the number of digits.

## Example

The variable contains a hexadecimal representation of a binary number that is stored in zoned decimal format in an IBM mainframe operating environment. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the input field.

```
data _null_;
   a=input('4EF1F2F3'x,s370fzds4.);
   b=input('60F1F2F3'x,s370fzds4.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=123 b=-123
```

# S370FZDTw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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* in other operating environments to read zoned decimal data from IBM mainframe files.

# Comparisons

- Zoned decimal, separate trailing-sign data is 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 *n* is the number of digits.

# Example

The variable contains a hexadecimal representation of a binary number that is stored in zoned decimal format in an IBM mainframe operating environment. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the input field.

```
data _null_;
   a=input('F1F2F34E'x,s370fzdt4.);
   b=input('F1F2F360'x,s370fzdt4.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=123 b=-123
```

# S370FZDUw.d Informat

Reads unsigned zoned decimal data in IBM mainframe format.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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* in 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 *n* is the number of digits.

## Example

The variable contains a hexadecimal representation of a binary number that is stored in zoned decimal format in an IBM mainframe operating environment. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one column of the input field.

```
data _null_;
   a=input('F1F2F3'x,s370fzdu3.);
   put a=;
run;
```

SAS writes the following to the log:

```
a=123
```

# SHRSTAMPw. Informat

Reads date and time values of SHR records.

Categories: CAS
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 8 bytes of information: 4 bytes of date data that are followed by 4 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. Then, the SHRSTAMP*w*. format 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 *ccyydddF*hhmmssth. The letters are defined as follows:

*ccyy*
is the 2-byte representation of the year. The *cc* portion is the 1-byte representation of a two-digit integer that represents the century. The *yy* portion is the 1-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 1 1/2 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 1-byte representation of two digits that correspond to the hour of the day.

*mm*
is the 1-byte representation of two digits that correspond to minutes.

*ss*
is the 1-byte representation of two digits that correspond to seconds.

*th*
is the 1-byte representation of two digits that correspond to a 100th of a second.

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

# Example

```
data values;
   shrstamp=input('0218239F12403576'x,shrstamp8.);
   put shrstamp ;
   put shrstamp datetime.;
run;
```

| Variable[1] | Result |
|---|---|
| | `----+----1----+----2` |
| `put shrstamp ;` | `5006666435.8` |
| `put shrstamp datetime.;` | `27AUG18:12:40:36` |

1 The variable 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, 2018 12:40:36.

# SMFSTAMPw. Informat

Reads time and date values of SMF records.

Categories: CAS
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 8 bytes of information: 4 bytes of time data that are followed by 4 bytes of date data.

**Tip** The time portion of an SMF record is a 4-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. Then, the SMFSTAMP*w*. informat 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 1-byte representation of two digits that correspond to the century.

*yy*
is the 1-byte representation of two digits that correspond to the year.

*ddd*
is the 1 1/2 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, in any operating environment, integer binary time values and packed decimal date values from files that are created on an IBM mainframe.

## Example

```
data one;
   input x: $hex16.;
   y=input(x,smfstamp8.);
   put y;
   put y datetime16.;
   datalines;
0058DC0C0108200F
;
run;
```

| Variable[1] | Result |
|---|---|
| | `----+----1----+----2` |
| `put y ;` | `1532016635` |
| `put y datetime;` | `18JUL08:16:10:35` |

1 The variable 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, 2018, 4:10:35 PM.

# STIMERw. Informat

Reads time values and determines whether the values are hours, minutes, or seconds; reads the output of the STIMER system option.

Category: Time

Restriction: This informat is not supported in a DATA step that runs in CAS.

## Syntax

**STIMER***w*.

## Syntax Description

***w***

specifies the width of the input field.

**Default** 10

**Range** 1–32

# 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 these 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.

# Example

```
data one;
   input test stimer10.;
   put test=;
   datalines;
234
333.4444
1846989
83769.6
202
;
run;
```

SAS writes the following output to the log:

```
test=234
test=333.4444
test=1846989
test=83769.6
test=202
```

# TIMEw. 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.

Categories: CAS
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.

You must specify a value in the hours field when using the TIME informat. You do not have to specify a value in the hours field when using the STIMER informat.

# Example

```
data test;
   time='12:56';
   sastime=input(time,time.);
   put sastime ;
   put sastime time8.;
run;
```

SAS writes the following output to the log:

```
46560
12:56:00
```

# See Also

**Formats:**

- “HHMMw.d” on page 221
- “HOURw.d” on page 224
- “MMSSw.d” on page 240
- “TIMEw.d” on page 465

**Functions:**

- “HOUR” in *SAS Functions and CALL Routines: Reference*
- “MINUTE” in *SAS Functions and CALL Routines: Reference*
- “SECOND” in *SAS Functions and CALL Routines: Reference*
- “TIME” in *SAS Functions and CALL Routines: Reference*

**Informats:**

- “STIMERw.”

# TODSTAMPw. Informat

Reads an 8-byte time-of-day stamp.

Categories: CAS
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 returns an 8-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*. in other operating environments to read time-of-day values that are produced by an IBM mainframe.

## Example

This example uses a SAS datetime value that corresponds to September 9, 2018, 11:59:58.

```
data one;
   date='D4E7F43876B80000'x;
   dt2=input(date,todstamp8.);

   put dt2  /
       dt2 datetime20.;
run;
```

SAS writes the following output to the log:

```
1852113598
09SEP2018:11:59:58
```

# TRAILSGNw. Informat

Reads a trailing plus (+) or minus (–) sign.

Category: Numeric

Restriction: This informat is not supported in a DATA step that runs in CAS.

## 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

```
data _null_;
   a=input('1',trailsgn8.);
   b=input('1,000',trailsgn8.);
   c=input('1+',trailsgn8.);
   d=input('1-',trailsgn8.);
   e=input('1.2',trailsgn8.);
   f=input('1.2+',trailsgn8.);
   g=input('1.2-',trailsgn8.);
   put a= b= c= d= e= f= g=;
run;
```

SAS writes the following output to the log:

```
a=1 b=1000 c=1 d=-1 e=1.2 f=1.2 g=-1.2
```

# TUw. Informat

Reads timer units.

Categories: CAS
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 4-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 in other operating environments.

## Example

The variable is a hexadecimal representation of a binary, 4-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.

```
data _null_;
   a=input('8FC7A9BC'x,tu4.);
   put a=;
run;
```

SAS writes the following to the log:

```
a=62818.411563
```

# VAXRBw.d Informat

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

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## 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 in 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” on page 670

# VMSZNw.d Informat

Reads VMS and MicroFocus COBOL zoned numeric data.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

## Syntax

**VMSZN***w.d*

### Syntax Description

***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 |
| 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 |

# Example

```
data _null_;
   a=input('1234',vmszn4.);
   b=input('123t',vmszn4.);
   put a= b=;
run;
```

SAS writes the following to the log:

```
a=1234 b=-1234
```

# See Also

**Formats:**

- “VMSZNw.d” on page 474

**Informats:**

- “ZDw.d” on page 725

# w.d Informat

Reads standard numeric data.

Categories: CAS
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 contains 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

```
data test;
   value='123.456';
   value2=input(value,8.3);
   put value2 ;
run;
```

SAS writes the following output to the log:

```
123.456
```

# WEEKUw. 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

Restriction: This informat is not supported in a DATA step that runs in CAS.

## 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 by 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 0

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 2018.

| Width | Format | Example |
|---|---|---|
| 3–4 | Www | w01 |
| 5–6 | yyWww | 18W01 |
| 7–8 | yyWwwdd | 18W0101 |
| 9–10 | yyyyWwwdd | 2018W0101 |
| 11–200 | yyyy-Www-dd | 2018-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 the week.

# Example

The current year is 2018 in these examples.

```
data one;
   v=input('W01',weeku3.);
   w=input('18W01',weeku5.);
   x=input('18W0101',weeku7.);
   y=input('2018W0101',weeku9.);
   z=input('2018-W01-01',weeku11.);
   put v;
   put w;
   put x;
   put y;
   put z;
run;
```

| Statement | Result |
|---|---|
| | `----+----1` |
| `put v;` | `21555` |
| `put w;` | `21191` |
| `put x;` | `21191` |
| `put y;` | `21191` |
| `put z;` | `21191` |

## See Also

**Formats:**

- "WEEKUw." on page 483
- "WEEKVw." on page 485
- "WEEKWw." on page 487

**Functions:**

- "WEEK" in *SAS Functions and CALL Routines: Reference*

**Informats:**

- "WEEKVw." on page 713
- "WEEKWw." on page 716

# WEEKVw. Informat

Reads a value in the form of a week number within a year and returns a SAS date value by using the V algorithm.

Category: Date

Restriction: This informat is not supported in a DATA step that runs in CAS.

## 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 0 and a maximum value of 53. Weeks begin on a Monday, and week 1 of the year is the week that includes January 4 and the first Thursday of the year. If the first Monday of January is 2, 3, or 4, 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 2018.

| Width | Format | Example |
|---|---|---|
| 3–4 | Www | w01 |
| 5–6 | yyWww | 18W01 |
| 7–8 | yyWwwdd | 18W0101 |
| 9–10 | yyyyWwwdd | 2018W0101 |
| 11–200 | yyyy-Www-dd | 2018–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 1 of the year is the week that includes January 4 and the first Thursday of the year. If the first Monday of January is 2, 3, or 4, 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 the week.

# Example

The current year is 2018 in these examples.

```
data one;
   v=input('W01',weekv3.);
   w=input('18W01',weekv5.);
   x=input('18W0101',weekv7.);
   y=input('2018W0101',weekv9.);
   z=input('2018-W01-01',weekv11.);
   put v;
   put w;
   put x;
   put y;
   put z;
run;
```

| Statement | Result |
|---|---|
| | `----+----1` |
| `put v;` | 21549 |
| `put w;` | 21185 |
| `put x;` | 21185 |
| `put y;` | 21185 |
| `put z;` | 21185 |

# See Also

**Formats:**

- “WEEKUw.” on page 483
- “WEEKVw.” on page 485
- “WEEKWw.” on page 487

**Functions:**

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

**Informats:**

- “WEEKUw.” on page 711

# WEEKWw. Informat

Reads a value in the form of a week number within the year and returns a SAS date value by using the W algorithm.

Category: Date

Restriction: This informat is not supported in a DATA step that runs in CAS.

## 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. The W algorithm calculates the SAS date value by 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 0 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 2018.

| Width | Format | Example |
|---|---|---|
| 3–4 | Www | w01 |
| 5–6 | yyWww | 18W01 |
| 7–8 | yyWwwdd | 18W0101 |

| Width | Format | Example |
|---|---|---|
| 9–10 | yyyyWwwdd | 2018W0101 |
| 11–200 | yyyy-Www-dd | 2018–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 the 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 2018 in these examples.

```
data one;
   v=input('W01',weekw3.);
   w=input('18W01',weekw5.);
   x=input('18W0101',weekw7.);
   y=input('2018W0101',weekw9.);
   z=input('2018-W01-01',weekw11.);
   put v;
   put w;
   put x;
   put y;
   put z;
run;
```

| Statement | Result |
|---|---|
| | `----+----1` |
| `put v;` | `21556` |
| `put w;` | `21185` |
| `put x;` | `21185` |
| `put y;` | `21185` |

| Statement | Result |
| --- | --- |
| `put z;` | `21185` |

## See Also

**Formats:**

- "WEEKUw." on page 483
- "WEEKVw." on page 485
- "WEEKWw." on page 487

**Function:**

- "WEEK" in *SAS Functions and CALL Routines: Reference*

**Informats:**

- "WEEKUw." on page 711
- "WEEKVw." on page 713

# YMDDTTMw.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 two or four digits.

Category: Date and Time

Alignment: Right

Restriction: This informat is not supported in a DATA step that runs in CAS.

## 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.

- 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.

> **IMPORTANT** If a fraction of a second is specified, the decimal point can be represented only by a period and is required.

# 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* 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.

## Example

```
data one;
   input dt ymddttm24.;
   put dt;
   put dt datetime24.;
   datalines;
2018-03-16 11:23:07.4
2018 03 16 11 23 07.4
18.3.16/11:23
run;
```

SAS writes the following output to the log:

```
1836818587.4
      16MAR2018:11:23:07
1836818587.4
      16MAR2018:11:23:07
1836818580
      16MAR2018:11:23:00
```

## See Also

**Informats:**

- “DATETIMEw.” on page 618
- “MDYAMPMw.d” on page 647

# YYMMDDw. Informat

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

Categories: CAS
Date

## 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 these 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 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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

# Example

```
data test;
   date='2018/05/04';
   sasdate=input(date,yymmdd10.);
   put sasdate;
   put sasdate date9.;
run;
```

SAS writes the following output to the log:

```
21308
04MAY2018
```

## See Also

**Formats:**

- "DATEw." on page 169
- "DDMMYYw." on page 175
- "MMDDYYw." on page 235
- "YYMMDDw." on page 498

**Functions:**

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

**Informats:**

- "DATEw." on page 617
- "DDMMYYw." on page 621
- "MMDDYYw." on page 649

**System Options:**

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

# YYMMNw. Informat

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

Categories: CAS
Date

## Syntax

**YYMMN***w*.

## Syntax Description

***w***

specifies the width of the input field.

**Default** 6

**Range** 4–32

# Details

SAS reads date values in one of these 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 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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

# Example

```
data test;
   date='201805';
   sasdate=input(date,yymmn6.);
   put sasdate;
   put sasdate date9.;
run;
```

SAS writes the following output to the log:

```
21305
01MAY2018
```

# See Also

**Formats:**

- "DATEw." on page 169
- "DDMMYYw." on page 175
- "YYMMDDw." on page 498
- "YYMMw." on page 497
- "YYMONw." on page 505

**Functions:**

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

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

**Informats:**

- "DATEw." on page 617
- "DDMMYYw." on page 621
- "MMDDYYw." on page 649
- "YYMMDDw." on page 720

**System Options:**

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

# YYQw. Informat

Reads quarters of the year in the form *yy*Q*q* or *yyyy*Q*q*.

Categories: Date
CAS

## Syntax

**YYQ***w*.

## Syntax Description

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

**Default** 4 (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 these 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 part of the 100-year span that is defined by the YEARCUTOFF= system option.

## Example

```
data test;
   date='2018Q2';
   sasdate=input(date,yyq6.);
   put sasdate;
   put sasdate date9.;
run;
```

SAS writes the following output to the log:

```
21275
01APR2018
```

## See Also

**Functions:**

- "QTR" in *SAS Functions and CALL Routines: Reference*
- "YEAR" in *SAS Functions and CALL Routines: Reference*
- "YYQ" in *SAS Functions and CALL Routines: Reference*

**System Options:**

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

# ZDw.d Informat

Reads zoned decimal data.

Categories: CAS
Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

See: "ZD Informat: UNIX" in *SAS Companion for UNIX Environments*
"ZDw.d: 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 1 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 1 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

```
data _null_;
   a=input('123{', zd4.);
   put a=;
run;
```

SAS writes the following to the log:

```
a=1230
```

## See Also

**Informats**

- “w.d” on page 710
- “ZDVw.d” on page 728

# ZDBw.d Informat

Reads zoned decimal data in which zeros have been left blank.

Category: Numeric

Restriction: This informat is not supported in a DATA step that runs in CAS.

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

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 is produced in IBM 1410, 1401, and 1620 form, where 0s are left blank rather than being punched.

# Example

```
data _null_;
   a=input('123{', zdb4.);
   put a=;
run;
```

SAS writes the following output to the log:

```
a=1230
```

# ZDVw.d Informat

Reads and validates zoned decimal data.

Category: Numeric

Restrictions: This informat is not supported in a DATA step that runs in CAS.
This option does not perform validation on z/OS.

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement that uses the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

# 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 1 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, the ZDV*w.d* informat 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 that range 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

```
data _null_;
   a=input('123{', zdv4.);
```

```
    b=input('123}', zdv4.);
    put a= b=;
run;
```

SAS write the following to the log:

```
a=1230 b=-1230
```

# See Also

**Informats:**

- “w.d” on page 710
- “ZDw.d” on page 725