About This Book

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

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

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

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=). 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 <option(s)> <argument-2 <option(s)> ...>

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-option(s))>**

```
<value-range-set-1 <(picture-1-option(s))>
<value-range-set-2 <(picture-2-option(s))> ...>>;
```
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
element, 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';
```
About This Book
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.

B8601DXw.
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 $yyyymmdTthmmss+hhmm.

B8601LXw.
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 $yyyymmdTthmmss+-hhmm.

B8601TXw.
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 $hhmss+-hhmm.

E8601DXw.
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-ddTth:mm:ss+hh:mm.

E8601LXw.
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-ddTth:mm:ss+-hh:mm.
**E8601TXw.**
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`.

**ODDSRw.**
writes odds ratios.

---

### 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:

- `BESTDw,p`  
  `E8601TZw.d`  
  `S370FPDUw.d`
- `B8601DTw.d`  
  `HHMMw.d`  
  `S370FZDw.d`
- `B8601TMw.d`  
  `HOURw.d`  
  `S370FZDLw.d`
- `COMMw.d`  
  `MDYAMPMw.d`  
  `S370FZDSw.d`
- `Dw.d`  
  `MMSSw.d`  
  `S370FZDTw.d`
- `DATEAMPMw.d`  
  `NEGPARENw.d`  
  `S370FZDUw.d`
- `DATETIMEw.d`  
  `NUMXw.d`  
  `TIMEw.d`
- `DOLLARw.d`  
  `PDw.d`  
  `TIMEAMPM`
- `DOLLARXw.d`  
  `PERCENTw.d`  
  `TOD`
- `Ew.`  
  `PERCENTNWw.d`  
  `w.d`
- `E8601DTw.d`  
  `PVALUEw.d`  
  `ZDw.d`
- `E8601TMw.d`  
  `S370FPDw.d`

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

---

### New Informats

**SAS Viya 3.4** has added the `$UUIDw.` informat.

**$UUIDw.**
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:
<table>
<thead>
<tr>
<th>New Alias</th>
<th>Existing Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>B8601DXw. on page 524</td>
<td>B8601DZw.d on page 525</td>
</tr>
<tr>
<td>B8601LXw. on page 526</td>
<td>B8601DTw.d on page 523</td>
</tr>
<tr>
<td>B8601TXw. on page 528</td>
<td>B8601TZw.d on page 528</td>
</tr>
<tr>
<td>E8601DXw. on page 546</td>
<td>E8601DZw.d on page 546</td>
</tr>
<tr>
<td>E8601LXw. on page 548</td>
<td>E8601DTw.d on page 544</td>
</tr>
<tr>
<td>E8601TXw. on page 552</td>
<td>E8601TZw.d on page 552</td>
</tr>
</tbody>
</table>
xiv  SAS Formats and Informats
Part 1

SAS Formats

Chapter 1
   About Formats ........................................... 3

Chapter 2
   Dictionary of Formats .................................. 55
Chapter 1

About Formats

Definition of Formats ............................................................... 4
Syntax .................................................................................. 4
Using Formats ........................................................................ 5
  Ways to Specify Formats ......................................................... 5
  Permanent versus Temporary Association ............................... 7
  User-Defined Formats ............................................................ 7
  Restrictions on Formatting Dates ............................................. 8
About SAS Date, Time, and Datetime Values ............................... 8
  Creating a SAS Date ............................................................... 8
  Definitions ........................................................................... 9
  Julian Date Formats and Astronomical Dates ............................ 9
  Two-Digit and Four-Digit Years ............................................. 12
  Five-Digit Years .................................................................. 13
  The Year 2000 ..................................................................... 13
  Working with SAS Dates and Times ....................................... 15
  Examples ............................................................................. 21
About Date and Time Intervals .................................................... 23
  Definitions ........................................................................... 23
  Syntax .................................................................................. 24
  Intervals by Category ............................................................ 24
  Example: Calculating a Duration .......................................... 26
  Boundaries of Intervals .......................................................... 27
  Single-Unit Intervals ............................................................. 27
  Multi-Unit Intervals ............................................................... 28
  Shifted Intervals .................................................................. 30
  Custom Intervals .................................................................. 31
  Retail Calendar Intervals: ISO 8601 Compliant ....................... 31
Byte Ordering for Integer Binary Data on Big Endian and
Little Endian Platforms ............................................................. 33
  Definitions ........................................................................... 33
  How Bytes Are Ordered Differently ....................................... 33
  Writing Data Generated on Big Endian and Little Endian Platforms 33
  Integer Binary Notation and Different Programming Languages 34
Data Conversions and Encodings ................................................. 35
Working with Packed Decimal and Zoned Decimal Data ............... 35
  Definitions ........................................................................... 35
  Types of Data ...................................................................... 36
**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. For example, the WORDS22. format, which converts numeric values to their equivalent in words, writes the numeric value 692 as *six hundred ninety-two*.

**Syntax**

SAS formats have the following form:

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

$  
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:

\[
\text{amount}=1145.32;
\text{put amount dollar10.2;}
\]

The DOLLAR\( w.d \) format in the PUT statement produces this result:

\$1,145.32

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

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

```sas
%macro tst(amount);
  %put %sysfunc(putn(&amount,dollar10.2));
%mend tst;
%tst (1154.23);
```

For more information, see “%SYSFUNC and %QSYSFUNC Functions” in SAS Macro Language: Reference.

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

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

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

For more information, see “FORMAT Statement” in SAS 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 Statement, 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 COMMAw.d format with the variables SALES1 through SALES3:

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

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

For more information, see “ATTRIB Statement” in SAS 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 “Names in the SAS Language” in *SAS Language Reference: Concepts* 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:

<table>
<thead>
<tr>
<th>System Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>FMTERR</td>
<td>SAS produces an error that causes the current DATA or PROC step to stop processing.</td>
</tr>
<tr>
<td>NOFMTERR</td>
<td>SAS continues processing and substitutes a default format, usually the BESTw. or $w. format. SAS also writes a note to the log indicating that the specified format was not found or could not be loaded.</td>
</tr>
</tbody>
</table>

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.

Restrictions on Formatting Dates

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. In SAS, the years 4000 and 8000 are not 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.

About SAS Date, Time, and Datetime Values

Creating a SAS Date

You can use these methods to create a SAS date:

• Read a value into SAS with an informat.
• Apply a format to an existing value.
• Use a date function.
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 ranging from A.D. November 1582 to A.D. 19,900. 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.
- 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 representing the number of seconds since midnight of the current day. SAS time values are between 0 and 86400.

SAS datetime value
is a value representing 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**

Calendar Date

SAS Date Values

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. They apply a Julian date format to a SAS date.

<table>
<thead>
<tr>
<th>Formats</th>
<th>Informats</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>JULDAY</td>
<td>JULIAN</td>
<td>DATEJUL</td>
</tr>
</tbody>
</table>
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.

**Example: Performing Calculations on Dates That Have Different Formats**
The following example performs these tasks:
- Creates a data set that contains SAS dates.
- Converts the dates into the MMDDYY10 format and the Julian format.
- Performs calculations on the two sets of dates, even though they have different formats.

```sas
data dates; /*  */
   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 mmdy10.
    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;

1 Create the data set of SAS dates.
2 Convert the dates into 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.
Output 1.1 Converting SAS Dates and Using the Results in Calculations

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 at the moment. To correctly handle two-digit years representing dates between 2000 and 2099, you should specify an appropriate YEARCUTOFF= value between 1901 and 2000. For more information, see the “YEARCUTOFF= System Option” in SAS System Options: Reference.

**Five-Digit Years**

Although some formats that specify a width large enough to accommodate formatting a five-digit year, such as DATETIME20., the 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.

```sas
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
formatted 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.

```sas
options nodate;

data schedule;
   input @1 jobid $ @6 projdate mmddyy10.;
data lines;
   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.;
```

*Example:* 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. The PUT statement writes the following lines to the SAS log:

SAS date=15639
formatted date=26OCT2002

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

The resulting output from the PROC PRINT statement looks like this:

**Output 1.2  Output Showing Four-Digit Years That Result from Setting YEARCUTOFF= to 1926**

<table>
<thead>
<tr>
<th>Obs</th>
<th>jobid</th>
<th>prodate</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A100</td>
<td>01/15/2025</td>
</tr>
<tr>
<td>2</td>
<td>A110</td>
<td>03/15/2025</td>
</tr>
<tr>
<td>3</td>
<td>A200</td>
<td>01/30/1996</td>
</tr>
<tr>
<td>4</td>
<td>B100</td>
<td>02/05/2012</td>
</tr>
<tr>
<td>5</td>
<td>B200</td>
<td>06/15/2012</td>
</tr>
</tbody>
</table>

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 show results that are similar to the first and second records. The fourth record specifies a two-digit year of 12, and the fifth one 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; starting with SAS 9.4, the default value is 1926.

For more information about how SAS handles dates, see the section on dates, times, and datetime values.

**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 coming into your SAS process to make sure that any dates containing two-digit years are correctly interpreted by the YEARCUTOFF= system option. Look out for the following situations:
• two-digit years that are distributed over more than a 100-year period. For dates covering more than a 100-year span, you must either use four-digit years in the data, or use 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.

• Make sure that output SAS data sets represent dates as SAS date values.

• Check your SAS programs to make sure 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, recognized by SAS, such as a time or date value, 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, which 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

<table>
<thead>
<tr>
<th>Task</th>
<th>Type of Language Element</th>
<th>Language Element</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write SAS date values in recognizable forms</td>
<td>Date formats</td>
<td>DATE.</td>
<td>19434</td>
<td>17MAR13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATE9.</td>
<td>19434</td>
<td>17MAR2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DAY.</td>
<td>19434</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYY.</td>
<td>19434</td>
<td>17/03/13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYY10.</td>
<td>19434</td>
<td>17/03/2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYB.</td>
<td>19434</td>
<td>17 03 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYB10.</td>
<td>19434</td>
<td>17 03 2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYC.</td>
<td>19434</td>
<td>17:03:13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYC10.</td>
<td>19434</td>
<td>17:03:2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYD.</td>
<td>19434</td>
<td>17-03-13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYD10.</td>
<td>19434</td>
<td>17-03-2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYN.</td>
<td>19434</td>
<td>17032013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYN6.</td>
<td>19434</td>
<td>170313</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYP.</td>
<td>19434</td>
<td>17.03.13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYP10.</td>
<td>19434</td>
<td>17.03.2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYYS.</td>
<td>19434</td>
<td>17/03/13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYY.</td>
<td>19434</td>
<td>03/17/13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYY10.</td>
<td>19434</td>
<td>03/17/2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYYB.</td>
<td>19434</td>
<td>03 17 13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYYB10.</td>
<td>19434</td>
<td>03 17 2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYYC.</td>
<td>19434</td>
<td>03:17:13</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYYC10.</td>
<td>19434</td>
<td>03:17:2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYYD.</td>
<td>19434</td>
<td>03-17-13</td>
</tr>
</tbody>
</table>

* JULDAY. * | 19434 | 76 |
* JULIAN. * | 19434 | 13076 |
<table>
<thead>
<tr>
<th>Task</th>
<th>Type of Language Element</th>
<th>Language Element</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write SAS date values in recognizable forms</td>
<td>Date formats</td>
<td>MMDDYYD10.</td>
<td>19434</td>
<td>03-17-2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMDDYYN.</td>
<td>19434</td>
<td>03172013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMDDYYN8.</td>
<td>19434</td>
<td>03172013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMDDYYP.</td>
<td>19434</td>
<td>03.17.13</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMDDYYP10.</td>
<td>19434</td>
<td>03.17.2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMDDYYS.</td>
<td>19434</td>
<td>03/17/13</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMDDYYS10.</td>
<td>19434</td>
<td>03/17/2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMYY.</td>
<td>19434</td>
<td>03M2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMYYC.</td>
<td>19434</td>
<td>03:2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMYYD.</td>
<td>19434</td>
<td>03-2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMYYN.</td>
<td>19434</td>
<td>032013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMYYP.</td>
<td>19434</td>
<td>03.2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MMYYS.</td>
<td>19434</td>
<td>03/2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MONNAME.</td>
<td>19434</td>
<td>March</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MONTH.</td>
<td>19434</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>MONYY.</td>
<td>19434</td>
<td>MAR13</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>PDJULG. *</td>
<td>19434</td>
<td>2013076F</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>PDJULL. *</td>
<td>19434</td>
<td>0100076F</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>WEEKDATE.</td>
<td>19434</td>
<td>Sunday, March 17, 2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>WEEKDAY.</td>
<td>19434</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>WORDDATE.</td>
<td>19434</td>
<td>March 17, 2013</td>
</tr>
<tr>
<td></td>
<td>Date formats</td>
<td>WORDDATX.</td>
<td>19434</td>
<td>17 March 2013</td>
</tr>
<tr>
<td>Quarter formats</td>
<td>Quarter formats</td>
<td>QTR.</td>
<td>19434</td>
<td>1</td>
</tr>
<tr>
<td>Quarter formats</td>
<td>Quarter formats</td>
<td>QTRR.</td>
<td>19434</td>
<td>1</td>
</tr>
<tr>
<td>Time formats</td>
<td>Time formats</td>
<td>TIME.</td>
<td>19434</td>
<td>5:23:54</td>
</tr>
<tr>
<td>Time formats</td>
<td>Time formats</td>
<td>TIMEAMPM.</td>
<td>19434</td>
<td>5:23:54 AM</td>
</tr>
<tr>
<td>Time formats</td>
<td>Time formats</td>
<td>TOD.</td>
<td>19434</td>
<td>05:23:54</td>
</tr>
</tbody>
</table>
Write SAS date values in recognizable forms

<table>
<thead>
<tr>
<th>Task</th>
<th>Type of Language Element</th>
<th>Language Element</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Year formats</td>
<td>YEAR.</td>
<td>19434</td>
<td>2013</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMM.</td>
<td>19434</td>
<td>2013M03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMMC.</td>
<td>19434</td>
<td>2013/03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMMD.</td>
<td>19434</td>
<td>2013-03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMMP.</td>
<td>19434</td>
<td>2013.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMMS.</td>
<td>19434</td>
<td>2013/03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMMN.</td>
<td>19434</td>
<td>201303</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMMDD.</td>
<td>19434</td>
<td>13-03-17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMON.</td>
<td>19434</td>
<td>2013MAR</td>
</tr>
<tr>
<td></td>
<td>Year/Quarter formats</td>
<td>YYQ.</td>
<td>19434</td>
<td>2013Q1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQC.</td>
<td>19434</td>
<td>2013:1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQD.</td>
<td>19434</td>
<td>2013-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQP.</td>
<td>19434</td>
<td>2013.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQS.</td>
<td>19434</td>
<td>2013/1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQN.</td>
<td>19434</td>
<td>20131</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQR.</td>
<td>19434</td>
<td>2013QI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQRC.</td>
<td>19434</td>
<td>2013:1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQRD.</td>
<td>19434</td>
<td>2013-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQRP.</td>
<td>19434</td>
<td>2013.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQRS.</td>
<td>19434</td>
<td>2013/1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQRN.</td>
<td>19434</td>
<td>2013I</td>
</tr>
</tbody>
</table>

* In SAS, a Julian date is a date in the form YYNNN or YYYYYNNN, 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

<table>
<thead>
<tr>
<th>Task</th>
<th>Type of Language Element</th>
<th>Language Element</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date Tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Task</td>
<td>Type of Language Element</td>
<td>Language Element</td>
<td>Input</td>
<td>Result</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------</td>
<td>-----------------</td>
<td>-------------</td>
<td>--------</td>
</tr>
<tr>
<td>Read calendar dates as SAS date</td>
<td>Date informats</td>
<td>DATE.</td>
<td>17MAR13</td>
<td>19434</td>
</tr>
<tr>
<td>Note: YEARCUTOFF=1926</td>
<td></td>
<td>DATE9.</td>
<td>17MAR2013</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYY.</td>
<td>170313</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DDMMYY8.</td>
<td>17032013</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JULIAN. *</td>
<td>13076</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JULIAN7. *</td>
<td>2013077</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYY.</td>
<td>031713</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MMDDYY8.</td>
<td>03172013</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MONYY.</td>
<td>MAR13</td>
<td>19418</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMMDD.</td>
<td>130317</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYMMDD8.</td>
<td>20130317</td>
<td>19434</td>
</tr>
<tr>
<td></td>
<td></td>
<td>YYQ.</td>
<td>13q1</td>
<td>19359</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DATETIME</td>
<td>17MAR2013 00:00:00</td>
<td>1679097600</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIME</td>
<td>14:45:32</td>
<td>53132</td>
</tr>
<tr>
<td>Return today's date as a SAS date value</td>
<td>Date functions</td>
<td>DATE() or TODAY() (equivalent)</td>
<td>( )</td>
<td>The SAS date value for today.</td>
</tr>
<tr>
<td>Extract calendar dates from SAS</td>
<td>Date functions</td>
<td>DAY</td>
<td>19434</td>
<td>17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HOUR</td>
<td>19434</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JULDATE *</td>
<td>19434</td>
<td>13076</td>
</tr>
<tr>
<td></td>
<td></td>
<td>JULDATE7 *</td>
<td>19434</td>
<td>2013076</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MINUTE</td>
<td>19434</td>
<td>23</td>
</tr>
<tr>
<td></td>
<td></td>
<td>MONTH</td>
<td>19434</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>QTR</td>
<td>19434</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SECOND</td>
<td>19434</td>
<td>54</td>
</tr>
<tr>
<td></td>
<td></td>
<td>WEEKDAY</td>
<td>19434</td>
<td>1</td>
</tr>
<tr>
<td>Task</td>
<td>Type of Language Element</td>
<td>Language Element</td>
<td>Input</td>
<td>Result</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>--------------------------</td>
<td>------------------</td>
<td>-----------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>Write a date as a constant in an expression</td>
<td>SAS date constant</td>
<td>'ddmmmyy'd</td>
<td>'17mar17'd</td>
<td>17mar17'd</td>
</tr>
<tr>
<td>Write today's date as a string</td>
<td>SYSDATE</td>
<td>SYSDATE9</td>
<td>&amp;SYSDATE9</td>
<td>The date at the time of SAS initialization in the form DDMMYY.</td>
</tr>
<tr>
<td>Write SAS time values as time values</td>
<td>time formats</td>
<td>HHMM.</td>
<td>19434</td>
<td>5:24</td>
</tr>
<tr>
<td>Read time values as SAS time values</td>
<td>Time informats</td>
<td>TIME.</td>
<td>05:23:54</td>
<td>19434</td>
</tr>
<tr>
<td>Write the current time as a string</td>
<td>SYSTIME</td>
<td>SYSTIME</td>
<td>&amp;SYSTIME</td>
<td>The time at the moment of execution, in the form HH:MM</td>
</tr>
<tr>
<td>Return the current time of day as a SAS time value</td>
<td>Time functions</td>
<td>TIME( )</td>
<td>( )</td>
<td>The SAS time value at moment of execution, in the form NNNNNN.NNN.</td>
</tr>
<tr>
<td>Return the time part of a SAS datetime value</td>
<td>Time functions</td>
<td>TIMEPART</td>
<td>17mar2013 05:11:43</td>
<td>5:11:43</td>
</tr>
<tr>
<td>Datetime Tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Write SAS datetime values as datetime values</td>
<td>Datetime formats</td>
<td>DATEAMPM</td>
<td>1679097600</td>
<td>17MAR13:12:00:00 AM</td>
</tr>
</tbody>
</table>
### Task

<table>
<thead>
<tr>
<th>Task</th>
<th>Type of Language Element</th>
<th>Language Element</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read datetime values as SAS datetime values</td>
<td>Datetime informats</td>
<td>DATETIME</td>
<td>1679097600</td>
<td>17MAR13:00:00:00</td>
</tr>
<tr>
<td>Return the current date and time of day as a SAS datetime value</td>
<td>Datetime functions</td>
<td>DATETIME()</td>
<td>()</td>
<td>The SAS datetime value at the moment of execution, in the form NNNNNNNNNNNN.N.</td>
</tr>
</tbody>
</table>

#### Interval Tasks

<table>
<thead>
<tr>
<th>Task</th>
<th>Type of Language Element</th>
<th>Language Element</th>
<th>Input</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return the number of specified time intervals that lie between the two date or datetime values</td>
<td>Interval functions</td>
<td>INTCK</td>
<td>week2 01aug60 01jan13</td>
<td>1368</td>
</tr>
<tr>
<td>Advances a date, time, or datetime value by a given interval, and returns a date, time, or datetime value</td>
<td>Interval functions</td>
<td>INTNX</td>
<td>day 17mar12 365</td>
<td>19434</td>
</tr>
</tbody>
</table>

* In SAS, a Julian date is a date in the form YYN or YYYYNNN, where Y 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 40 and “Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467.


### 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.

```sas
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 1.3  Datetime, Date, and Time Values for 86399

<table>
<thead>
<tr>
<th>Obs</th>
<th>Time1</th>
<th>Date1</th>
<th>Time2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01JAN60:23:59:59</td>
<td>20JUL2196</td>
<td>11:59:59 PM</td>
</tr>
</tbody>
</table>

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.

```sas
data meeting;
  input region $ mtg : mmddyy8.;
  sendmail=mtg-45;
datalines;
N 11-24-12
S 12-28-12
E 12-03-12
```

Chapter 1 • About Formats
About Date and Time Intervals

Definitions

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

TIP  
Date and datetimes 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 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 specified by the date in the beginning value, SAS treats it 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 them 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:

\[ \text{name}<\text{multiple}><\text{.starting-point}> \]

The terms in an interval have the following definitions:

name

is the name of the interval. See the following table for a list of intervals and their definitions.

multiple

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

 starters-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

\textbf{Table 1.3 Intervals Used with Date and Time Functions}

<table>
<thead>
<tr>
<th>Category</th>
<th>Interval</th>
<th>Definition</th>
<th>Default Starting Point</th>
<th>Shift Period</th>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>DAY</td>
<td>Daily intervals</td>
<td>Each day</td>
<td>Days</td>
<td>DAY3</td>
<td>Three-day intervals starting on Sunday</td>
</tr>
<tr>
<td></td>
<td>WEEK</td>
<td>Weekly intervals of seven days</td>
<td>Each Sunday</td>
<td>Days (1=Sunday … 7=Saturday)</td>
<td>WEEK.7</td>
<td>Weekly with Saturday as the first day of the week</td>
</tr>
<tr>
<td></td>
<td>WEEKDAY &lt;daysW&gt;</td>
<td>Daily intervals with Friday-Saturday-Sunday</td>
<td>Each day</td>
<td>Days</td>
<td>WEEKDAY1W</td>
<td>Six-day week with Sunday as a weekend day</td>
</tr>
<tr>
<td>Category</td>
<td>Interval</td>
<td>Definition</td>
<td>Default Starting Point</td>
<td>Shift Period</td>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>----------</td>
<td>------------</td>
<td>------------------------</td>
<td>--------------</td>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>counted as the same day (five-day work week with a Saturday-Sunday weekend). <em>days</em> identifies the weekend days by number (1=Sunday ... 7=Saturday). By default, <em>days</em>=17.</td>
<td></td>
<td></td>
<td></td>
<td>WEEKDAY35W</td>
<td>Five-day week with Tuesday and Thursday as weekend days (W indicates that day 3 and day 5 are weekend days)</td>
</tr>
<tr>
<td>TENDAY</td>
<td>Ten-day intervals (a U.S. automobile industry convention)</td>
<td>First, eleventh, and twenty-first of each month</td>
<td>Ten-day periods</td>
<td></td>
<td>TENDAY4.2</td>
<td>Four ten-day periods starting at the second TENDAY period</td>
</tr>
<tr>
<td>SEMIMONTH</td>
<td>Half-month intervals</td>
<td>First and sixteenth of each month</td>
<td>Semi-monthly periods</td>
<td></td>
<td>SEMIMONTH2.2</td>
<td>Intervals from the sixteenth of one month through the fifteenth of the next month</td>
</tr>
<tr>
<td>MONTH</td>
<td>Monthly intervals</td>
<td>First of each month</td>
<td>Months</td>
<td></td>
<td>MONTH2.2</td>
<td>February-March, April-May, June-July, August-September, October-November, and December-January of the following year</td>
</tr>
<tr>
<td>QTR</td>
<td>Quarterly (three-month) intervals</td>
<td>January 1, April 1, July 1, October 1</td>
<td>Months</td>
<td></td>
<td>QTR3.2</td>
<td>Three-month intervals starting on April 1, July 1, October 1, and January 1</td>
</tr>
<tr>
<td>Category</td>
<td>Interval</td>
<td>Definition</td>
<td>Default Starting Point</td>
<td>Shift Period</td>
<td>Example</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------</td>
<td>-------------------------------------------------</td>
<td>------------------------</td>
<td>--------------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>SEMIYEAR</td>
<td>Semiannual (six-month) intervals</td>
<td>January 1</td>
<td>Months</td>
<td>SEMIYEAR.3</td>
<td>Six-month intervals, March-August, and September-February</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>July 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>YEAR</td>
<td>Yearly intervals</td>
<td>January 1</td>
<td>Months</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date/time</td>
<td>Add DT to any of the date intervals</td>
<td>Interval corresponding to the associated date interval</td>
<td>Midnight of January 1, 1960</td>
<td></td>
<td>DTMONTH</td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>SECOND</td>
<td>Second intervals</td>
<td>Start of the day (midnight)</td>
<td>Seconds</td>
<td>DTWEEKDAY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MINUTE</td>
<td>Minute intervals</td>
<td>Start of the day (midnight)</td>
<td>Minutes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>HOUR</td>
<td>Hourly intervals</td>
<td>Start of the day (midnight)</td>
<td>Hours</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example: Calculating a Duration**

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

```plaintext
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;
```
Calculating the Duration between Start and End Dates

### 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

<table>
<thead>
<tr>
<th>Example</th>
<th>Results</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mnthnum1=intck('month', '25aug2000'd, '05sep2000'd);</code></td>
<td><code>mnthnum1=1</code></td>
<td>The number of MONTH intervals the INTCK function counts depends on whether the first day of a month falls within the period.</td>
</tr>
<tr>
<td><code>mnthnum2=intck('month', '01aug2000'd, '31aug2000'd);</code></td>
<td><code>mnthnum2=0</code></td>
<td></td>
</tr>
<tr>
<td><code>next=intnx('month', '25aug2000'd,1);</code></td>
<td><code>next represents 01sep2000</code></td>
<td>The INTNX function produces the SAS date value that corresponds to the beginning of the next interval.</td>
</tr>
</tbody>
</table>

*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

<table>
<thead>
<tr>
<th>Single-Unit Interval</th>
<th>Beginning Point on the Calendar</th>
</tr>
</thead>
<tbody>
<tr>
<td>DAY</td>
<td>each day</td>
</tr>
<tr>
<td>WEEKDAY</td>
<td>for a standard weekday</td>
</tr>
<tr>
<td></td>
<td>• Start day–End day</td>
</tr>
<tr>
<td></td>
<td>• Monday–Monday</td>
</tr>
<tr>
<td></td>
<td>• Tuesday–Tuesday</td>
</tr>
<tr>
<td></td>
<td>• Wednesday–Wednesday</td>
</tr>
<tr>
<td></td>
<td>• Thursday–Thursday</td>
</tr>
<tr>
<td></td>
<td>• Friday–Sunday</td>
</tr>
<tr>
<td>WEEK</td>
<td>each Sunday</td>
</tr>
<tr>
<td>TENDAY</td>
<td>the first, eleventh, and twenty-first of each month</td>
</tr>
<tr>
<td>SEMIMONTH</td>
<td>the first and sixteenth of each month</td>
</tr>
<tr>
<td>MONTH</td>
<td>the first of each month</td>
</tr>
<tr>
<td>QTR</td>
<td>the first of January, April, July, and October</td>
</tr>
<tr>
<td>SEMIYEAR</td>
<td>the first of January and July</td>
</tr>
<tr>
<td>YEAR</td>
<td>the first of January</td>
</tr>
</tbody>
</table>

Single-unit time intervals begin as follows:

Table 1.6  Single-Unit Time Intervals

<table>
<thead>
<tr>
<th>Single-Unit Time Intervals</th>
<th>Beginning Point</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECOND</td>
<td>each second</td>
</tr>
<tr>
<td>MINUTE</td>
<td>each minute</td>
</tr>
<tr>
<td>HOUR</td>
<td>each hour</td>
</tr>
</tbody>
</table>

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 beginning 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

<table>
<thead>
<tr>
<th>SAS statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>howmany1=intck('month2','15feb2000'd,'15mar2000'd);</td>
<td>howmany1=1</td>
</tr>
<tr>
<td>count=intck('day50','01oct1998'd,'01jan1999'd);</td>
<td>count=1</td>
</tr>
</tbody>
</table>

In the above 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, as follows:

Table 1.8  Using the INTNX Function

<table>
<thead>
<tr>
<th>SAS statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>start=intnx('day50','01oct98'd,1);</td>
<td>SAS date value 14200, or Nov 17, 1998</td>
</tr>
</tbody>
</table>

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:

Figure 1.2  Calculating Multi-Week Intervals

<table>
<thead>
<tr>
<th>Dec</th>
<th>Su</th>
<th>Mo</th>
<th>Tu</th>
<th>We</th>
<th>Th</th>
<th>Fr</th>
<th>Sa</th>
<th>Jan</th>
</tr>
</thead>
<tbody>
<tr>
<td>1959</td>
<td>27</td>
<td>28</td>
<td>29</td>
<td>30</td>
<td>31</td>
<td>1</td>
<td>2</td>
<td>1960</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>count=intck('week2', '01aug98'D, '31aug98'D);</td>
<td>count=3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin=intnx('week2', '01aug1998'd,1);</td>
<td>“Begin” represents SAS date 14093 or August 02, 1998</td>
</tr>
</tbody>
</table>

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 twenty-fifth 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

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>start=intnx('day50.5','01jan1960'd,1);</td>
<td>SAS date value 4, or Jan 5, 1960</td>
</tr>
</tbody>
</table>

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, it 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

<table>
<thead>
<tr>
<th>SAS Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>start=intnx('week2.8','01jan1960'd,1);</td>
<td>SAS date value 2, or Jan 3, 1960</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Interval</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YEARV</td>
<td>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 is written in ISO 8601 weeks (WEEKV).</td>
</tr>
<tr>
<td>R445YR</td>
<td>is the same as YEARV except that in the retail industry the beginning subperiod is 4-4-5 months (R445MON).</td>
</tr>
<tr>
<td>R454YR</td>
<td>is the same as YEARV except that in the retail industry the beginning subperiod is 4-5-4 months (R454MON).</td>
</tr>
<tr>
<td>R544YR</td>
<td>is the same as YEARV except that in the retail industry the beginning subperiod is 5-4-4 months (R544MON).</td>
</tr>
<tr>
<td>R445QTR</td>
<td>specifies retail 4-4-5 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The beginning subperiod is 4-4-5 months (R445MON).</td>
</tr>
<tr>
<td>R454QTR</td>
<td>specifies retail 4-5-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The beginning subperiod is 4-5-4 months (R454MON).</td>
</tr>
<tr>
<td>R544QTR</td>
<td>specifies retail 5-4-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The beginning subperiod is 5-4-4 months (R544MON).</td>
</tr>
<tr>
<td>R445MON</td>
<td>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 is 4-4-5 months (R445MON).</td>
</tr>
<tr>
<td>R454MON</td>
<td>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 is 4-5-4 months (R454MON).</td>
</tr>
<tr>
<td>R544MON</td>
<td>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 is 5-4-4 months (R544MON).</td>
</tr>
</tbody>
</table>
Interval | Description
--- | ---
WEEKV | specifies ISO 8601 weekly intervals of seven days. Each week begins on Monday. The beginning subperiod 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, 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.
<table>
<thead>
<tr>
<th>Platform for Which the Data Was Created</th>
<th>Platform That Writes the Data</th>
<th>Signed Integer</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>big endian</td>
<td>big endian</td>
<td>yes</td>
<td>IB or S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>big endian</td>
<td>no</td>
<td>PIB, S370FPIB, S370FIBU</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>yes</td>
<td>S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>no</td>
<td>S370FPIB</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>no</td>
<td>PIBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>yes</td>
<td>IB or IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>no</td>
<td>PIB or PIBR</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>yes</td>
<td>S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>no</td>
<td>S370FPIB</td>
</tr>
<tr>
<td>little endian</td>
<td>either</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>either</td>
<td>no</td>
<td>PIBR</td>
</tr>
</tbody>
</table>

**Integer Binary Notation and Different Programming Languages**

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

<table>
<thead>
<tr>
<th>Language</th>
<th>2 Bytes</th>
<th>4 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I</td>
<td>FIXED BIN(15)</td>
<td>FIXED BIN(31)</td>
</tr>
<tr>
<td>Fortran</td>
<td>INTEGER*2</td>
<td>INTEGER*4</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP PIC 9(4)</td>
<td>COMP PIC 9(8)</td>
</tr>
<tr>
<td>IBM assembler</td>
<td>H</td>
<td>F</td>
</tr>
<tr>
<td>C</td>
<td>short</td>
<td>long</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>IBM versus COBOL II Clauses</th>
<th>Corresponding S370Fxxx Formats and Informats</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC S9(X) PACKED-DECIMAL</td>
<td>S370FPDw.</td>
</tr>
<tr>
<td>PIC 9(X) PACKED-DECIMAL</td>
<td>S370FPDUw.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY</td>
<td>S370FZDw.</td>
</tr>
<tr>
<td>PIC 9(W) DISPLAY</td>
<td>S370FZDUw.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN LEADING</td>
<td>S370FZDLw.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN LEADING SEPARATE</td>
<td>S370FZDSw.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN TRAILING SEPARATE</td>
<td>S370FZDTw.</td>
</tr>
</tbody>
</table>

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 \(\text{ceil}(\frac{x+1}{2})\). For PIC 9(X) PACKED-DECIMAL, W is \(\text{ceil}(\frac{x}{2})\). For example, PIC S9(5) PACKED-DECIMAL represents five digits. If a sign is included, six nibbles are needed. \(\text{ceil}(\frac{5+1}{2})\) 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:
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 Informs**

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.

<table>
<thead>
<tr>
<th>Format</th>
<th>Type of Data Representation</th>
<th>Corresponding Informat</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD</td>
<td>Packed decimal</td>
<td>PD</td>
<td>Local, signed packed decimal.</td>
</tr>
<tr>
<td>PK</td>
<td>Packed decimal</td>
<td>PK</td>
<td>Unsigned packed decimal; not specific to your operating environment.</td>
</tr>
<tr>
<td>ZD</td>
<td>Zoned decimal</td>
<td>ZD</td>
<td>Local zoned decimal.</td>
</tr>
<tr>
<td>none</td>
<td>Zoned decimal</td>
<td>ZDB</td>
<td>Translates EBCDIC blank (hexadecimal 40) to EBCDIC zero (hexadecimal F0); corresponds to the informat as zoned decimal.</td>
</tr>
<tr>
<td>none</td>
<td>Zoned decimal</td>
<td>ZDV</td>
<td>Zoned decimal notation other than IBM.</td>
</tr>
<tr>
<td>S370FPD</td>
<td>Packed decimal</td>
<td>S370FPD</td>
<td>Last nibble C (positive) or D (negative).</td>
</tr>
<tr>
<td>S370FPDU</td>
<td>Packed decimal</td>
<td>S370FPDU</td>
<td>Last nibble always F (positive).</td>
</tr>
<tr>
<td>S370FZD</td>
<td>Zoned decimal</td>
<td>S370FZD</td>
<td>Last byte contains sign in upper nibble: C (positive) or D (negative).</td>
</tr>
<tr>
<td>S370FZDU</td>
<td>Zoned decimal</td>
<td>S370FZDU</td>
<td>Unsigned; sign nibble always F.</td>
</tr>
<tr>
<td>Format</td>
<td>Type of Data Representation</td>
<td>Corresponding Informat</td>
<td>Comment</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------</td>
<td>------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>S370FZDL</td>
<td>Zoned decimal</td>
<td>S370FZDL</td>
<td>Sign nibble in first byte in informat; separate leading sign byte of hexadecimal C0 (positive) or D0 (negative) in format.</td>
</tr>
<tr>
<td>S370FZDS</td>
<td>Zoned decimal</td>
<td>S370FZDS</td>
<td>Leading sign of – (hexadecimal 60) or + (hexadecimal 4E).</td>
</tr>
<tr>
<td>S370FZDT</td>
<td>Zoned decimal</td>
<td>S370FZDT</td>
<td>Trailing sign of – (hexadecimal 60) or + (hexadecimal 4E).</td>
</tr>
<tr>
<td>PDJULI</td>
<td>Packed decimal</td>
<td>PDJULI</td>
<td>Julian date in packed representation - IBM computation.</td>
</tr>
<tr>
<td>PDJULG</td>
<td>Packed decimal</td>
<td>PDJULG</td>
<td>Julian date in packed representation - Gregorian computation.</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>RMFDUR</td>
<td>Input layout is mmsstttF.</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>SHRSTAMP</td>
<td>Input layout is yyyydddFhhmmsssth, where yyyydddF is the packed Julian date; yyyy is a zero-based year from 1900.</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>SMFSTAMP</td>
<td>Input layout is xxxxxxxxyyyydddF, where yyyydddF is the packed Julian date; yyyy is a zero-based year from 1900.</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>PDTIME</td>
<td>Input layout is 0hhmmssF.</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>RMFSTAMP</td>
<td>Input layout is 0hhmmssFyyyydddF, where yyyydddF is the packed Julian date; yyyy is a zero-based year from 1900.</td>
</tr>
</tbody>
</table>
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 informatss 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= System Option” 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 yyyymmddThhmmss+|–hhmm or the ISO 8601 extended form yyyy-mm-ddThh: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:

- 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:

<table>
<thead>
<tr>
<th>ISO 8601 Notation</th>
<th>Example</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>yyyymmd</td>
<td>20180915</td>
</tr>
<tr>
<td>Writes the date from a datetime.</td>
<td>yyyymmd</td>
<td>20180915</td>
</tr>
<tr>
<td>Time</td>
<td>hhmmssffffff</td>
<td>155300322348</td>
</tr>
</tbody>
</table>

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.
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-ddThh:mm:ss<ffffff>`

Time with time zone: `hh:mm:ss<ffffff>±–hh:mm <ffffff>±–hh:mm:ss<ffffff>Z`

Datetime with time zone: `yyyy-mm-ddThh:mm:ss<ffffff>±–hh:mm or yyyy-mm-ddThh: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 171
  Writes the date from a datetime. | yyyy-mm-dd | 2018-09-15 | E8601DNw. on page 132
**Time** | hh:mm:ss.ffffff | 15:53:00.322348 | E8601TMw.d on page 141
  Zero meridian time and time zone offset. | hh:mm:ss.ffffff | 15:53:00+00:00 | E8601TZw.d on page 143
  The time zone offset is always +00:00.
  Zero meridian time that uses a local time zone offset. | hh:mm:ss.ffffff | 15:53:00+05:00 | E8601LZw.d on page 139
  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 182
**Datetime** | yyyy-mm-ddThh:mm:ss.ffffff | 2018-09-15T15:53:00 | E8601DTw.d on page 133
  Zero meridian datetime and time zone offset. | yyyy-mm-ddThh:mm:ss.ffffff | 2018-09-15T15:53:00+00:00 | E8601DZw.d on page 176
  The time zone offset is always +00:00.
  Zero meridian datetime that uses a time zone offset for the user local time. | yyyy-mm-ddThh:mm:ss+|–|hh:mm | 2018-09-15T15:53:00–05:00 | E8601LXw. on page 178
  Converts a datetime value to the user local time by using a time zone offset for the user local time. | yyyy-mm-ddThh:mm:ss+|–|hh:mm | 2018-09-15T185300–0500 | E8601DXw. on page 175

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


### 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:

<table>
<thead>
<tr>
<th>TZ or DZ</th>
<th>LZ</th>
<th>LX</th>
<th>TX or DX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero meridian date or datetime that uses a zero meridian time zone offset</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero meridian time that uses a local time zone offset</td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Zero meridian datetime that uses a user local time zone offset</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>User local time or datetime that uses a user local time zone offset</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

**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.

```sas
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.;
```
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= 2018-09-14T22:53:00-05:00
tm= 5:53:00
e8601tz=05:53:00+00:00
e8601lz=05:53:00-05:00
e8601tx=21:53:00-08:00
```

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 B8601DAw. and E8601DAw. informats. You can write dates in the same form by using the B8601DAw. and E8601DAw. 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 b8601da$ +1 var2 e8601da10.;
put var1=b8601da. var2=e8601da.;
datelines;
20180402 2018-04-08
;
run;

Here is the output from the SAS log:

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

By using the B8601DNw. and E8601DNw. formats, you can extrapolate the date from a datetime value. This example reads the datetime value by using the B8601DNw. informat, and writes the date by using the B8601DNw. format:

```
data _null_;  
input @1 dt b8601dn.;  
put dt b8601dn.;  
datelines;  
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 B8601TMw.d and E8601TMw.d informats. The B8601TMw.d and the E8601TMw.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 E8601TZw.d informat to read a time value that contains a time zone offset. The B8601TZw.d and E8601TZw.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:
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 B8601TXw. or E8601TXw. 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 B8601LZw. format or the E8601LZw. 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 B8601DTw. and E8601DTw. informats and writes the datetime values by using the B8601DTw. and E8601DTw. 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 B8601DTw.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 B8601DZw. and E8601DZw. 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:

```sas
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 B8601DXXw. and E8601DXXw. 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 E8601DZw. informat and writes the datetime value for the time zone in Zurich (+02:00):

```sas
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 E8601DXXw. format. The time zone offset is the difference between the America/Anchorage time zone and UTC, which is nine hours.

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

Here is the output from the SAS log:
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 B8601LZw. format or the E8601LZw. 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:

```sas
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 B8601LXw. or E8601LXw. format. This example writes the datetime for a user by using the time zone name PST (Pacific Time):

```sas
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:

- \(PyyyymmddThhmmss\).
- \(Pyyyy-mm-ddThh:mm:ss\).
- \(Pyyyy-mm-dd\) is a span of years, months, and days.
- \(PnYnMnDTnHnMnS\) is the number of years (Y), months (M), days (D), hours (H), minutes (M), and seconds (S).
- \(PnW\) 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 \( P_nW \) 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:

- \( \text{datetime/datetime} \)
- \( \text{datetime/duration} \)
- \( \text{duration/datetime} \)

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

<table>
<thead>
<tr>
<th>Time Component</th>
<th>ISO 8601 Notation</th>
<th>Example</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration - Basic Notation</td>
<td>PyyyyymmddTthmmssfff</td>
<td>P20180915T155300</td>
<td>$N8601BA</td>
</tr>
<tr>
<td></td>
<td>–PyyyyymmddTthmmssfff</td>
<td>–P20180915T155300</td>
<td>$N8601BA</td>
</tr>
<tr>
<td>Duration - Extended Notation</td>
<td>Pyyyy-mm-ddTth:mm:ss.fff</td>
<td>P2018-09-15T15:53:00</td>
<td>$N8601EA</td>
</tr>
<tr>
<td></td>
<td>–Pyyyy-mm-ddTth:mm:ss.fff</td>
<td>–P2018-09-15T15:53:00</td>
<td>$N8601EA</td>
</tr>
<tr>
<td>Duration - Basic and Extended Notation</td>
<td>PnYnMoDTnHnMnS</td>
<td>P2y10m14dT20h13m45s</td>
<td>$N8601B</td>
</tr>
<tr>
<td></td>
<td>–PnYnMoDTnHnMnS</td>
<td>–P2y10m14dT20h13m45s</td>
<td>$N8601B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$N8601E</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$N8601E</td>
</tr>
<tr>
<td>PnW (weeks)</td>
<td>P6w</td>
<td></td>
<td>$N8601B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$N8601E</td>
</tr>
<tr>
<td>Interval - Basic Notation</td>
<td>yyyyymddTthmmssfff / yyyyymddTthmmssfff</td>
<td>20170915T155300/20181113T000000</td>
<td>$N8601BA</td>
</tr>
<tr>
<td></td>
<td>PnYnMoDTnHnMnS / yyyyymddTthmmssfff</td>
<td>P2y10M14dT20h13m45s/20180915T155300</td>
<td>$N8601B</td>
</tr>
<tr>
<td></td>
<td>yyyyymmddTthmmssfff / PnYnMoDTnHnMnS</td>
<td>20180915T155300/20180915T155300</td>
<td>$N8601BA</td>
</tr>
<tr>
<td>Interval - Extended Notation</td>
<td>yyyy-mm-ddTth:mm:ss.fff / yyyy-mm-ddTth:mm:ss.fff</td>
<td>2017-09-15T15:53:00/2018-09-15T15:53:00</td>
<td>$N8601EA</td>
</tr>
<tr>
<td></td>
<td>PnYnMoDTnHnMnS/ yyyy-mm-ddTth:mm:ss.fff</td>
<td>P2y10M14dT20h13m45s/2018-09-15T15:53:00</td>
<td>$N8601E</td>
</tr>
</tbody>
</table>
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 \( PnYnMnDTnHnMnS \) 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:

<table>
<thead>
<tr>
<th>Format</th>
<th>Datetime Form</th>
<th>Duration Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N8601H</td>
<td>yyyy-mm-ddThh:mm:ss</td>
<td>( PnYnMnDTnHnMnS )</td>
<td>--09-15T15::00</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>P2y10M14dT20h13m45s</td>
</tr>
</tbody>
</table>
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-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:

<table>
<thead>
<tr>
<th>Duration</th>
<th>Extended Normalized Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>p3y13m</td>
<td>p0004-01</td>
</tr>
<tr>
<td>pt24h24m65s</td>
<td>P---01T::25:05</td>
</tr>
</tbody>
</table>
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, \texttt{pT12:60:13} becomes \texttt{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:

- \texttt{201209.5}
- \texttt{P2012-09-15T10.33}
- \texttt{2012-09-15/P0003-03-03,333}
Chapter 2

Dictionary of Formats

Formats Documented in Other Publications ........................................ 60
Formats by Category ................................................................. 60
Dictionary ................................................................. 93
$ASCIIw. Format .............................................................. 93
$BASE64Xw. Format .......................................................... 94
$BINARYw. Format ............................................................ 95
$CHARw. Format .............................................................. 96
$CSTR Format ............................................................... 97
$EBCDICw. Format ............................................................ 98
$HEXw. Format ............................................................... 100
$MSGCASEw. Format ....................................................... 101
$N8601Bw.d Format ...................................................... 102
$N8601Aw.d Format ...................................................... 104
$N8601Ew.d Format ...................................................... 106
$N8601Aw.d Format ...................................................... 107
$N8601EHw.d Format ..................................................... 109
$N8601EXw.d Format ..................................................... 111
$N8601Hw.d Format ...................................................... 113
$N8601Xw.d Format ...................................................... 115
$OCTALw. Format .......................................................... 116
$QUOTEw. Format .......................................................... 117
$REVERJw. Format ........................................................ 119
$REVERSw. Format ....................................................... 120
$UPCASEw. Format ........................................................ 121
$UUIDw. Format ........................................................... 122
$VARYINGw. Format ...................................................... 123
$w. Format ................................................................. 125
BESTw. Format ............................................................ 126
BESTDw.p Format ........................................................ 128
BINARYw. Format .......................................................... 130
B8601Aw. Format ........................................................ 131
B8601Nw. Format ........................................................ 132
B8601Tw.d Format ...................................................... 133
B8601Xw. Format ........................................................ 135
B8601Zw. Format ........................................................ 136
B8601LXw. Format ........................................................ 138
B8601LZw. Format ........................................................ 139
B8601TMw.d Format ...................................................... 141
B8601TXw. Format ........................................................ 142
B8601TZw. Format ........................................................ 143
Chapter 2 • Dictionary of Formats

COMMAw.d Format ......................................................... 145
COMMAXw.d Format ................................................. 147
Dw.p Format ............................................................. 148
DATEw. Format .......................................................... 150
DATEAMPw.d Format ................................................. 151
DATETIMEw.d Format ............................................... 153
DAYw. Format .......................................................... 155
DDMMYYw. Format .................................................. 156
DDMMYYxw. Format ............................................... 158
DOLLARw.d Format .................................................... 160
DOLLARXw.d Format ................................................... 161
DOWNAMEw. Format ............................................... 163
DTDATEw. Format ...................................................... 164
DTMONYYw. Format .................................................. 165
DTWKRDATEXw. Format ............................................. 166
DTYEARw. Format ..................................................... 168
DTYEYCw. Format ..................................................... 169
Ew. Format .............................................................. 170
E8601DAw. Format .................................................... 171
E8601DNw. Format ..................................................... 172
E8601DTw.d Format .................................................... 173
E8601DXw. Format ..................................................... 175
E8601DZw. Format ..................................................... 176
E8601LXw. Format ..................................................... 178
E8601LZw. Format ..................................................... 179
E8601TMw.d Format .................................................... 181
E8601TXw. Format ..................................................... 182
EUROw.d Format ....................................................... 186
EUROXw.d Format ..................................................... 189
FLOATw.d Format ....................................................... 191
FRACTw. Format ....................................................... 192
HEXw. Format .......................................................... 193
HHMMw.d Format ..................................................... 195
HOURw.d Format ....................................................... 197
IBw.d Format .......................................................... 198
IBRWw.d Format ....................................................... 200
IEEEw.d Format ....................................................... 201
JULDAYw. Format ..................................................... 202
JULIANw. Format ....................................................... 204
MDYAMPMw.d Format ................................................. 205
MMDDYYw. Format ................................................... 207
MMDDYYxw. Format .................................................. 209
MMSSw.d Format ....................................................... 211
MMYYw. Format ....................................................... 212
MMYYxw. Format ..................................................... 214
MONNAMEw. Format .................................................. 216
MONTHw. Format ....................................................... 217
MONYYw. Format ....................................................... 218
NEGPARENw.d Format ................................................ 219
NENGow. Format ...................................................... 221
NLBESTw. Format ..................................................... 222
NLDATEw. Format ..................................................... 224
NLDATELw. Format .................................................... 225
NLDATEmw. Format ................................................... 226
NLDATEMDw. Format ................................................ 227
NLMNIGBPw.d Format ........................................... 283
NLMNIHKDw.d Format ........................................... 284
NLMNIHRKw.d Format ........................................... 285
NLMNIIUFw.d Format ........................................... 286
NLMNIIDRw.d Format ........................................... 287
NLMNIILSw.d Format ........................................... 288
NLMNIINRw.d Format ........................................... 289
NLMNIJPYw.d Format ........................................... 290
NLMNIKRWw.d Format ........................................... 291
NLMNILTLw.d Format ........................................... 292
NLMNILVLw.d Format ........................................... 293
NLMNIMOPw.d Format ........................................... 294
NLMNIMXNw.d Format ........................................... 295
NLMNIMYRw.d Format ........................................... 296
NLMNINO Kw.d Format .......................................... 297
NLMNINZDw.d Format .......................................... 298
NLMNIPLNw.d Format .......................................... 299
NLMNIRUBw.d Format .......................................... 300
NLMNISEKw.d Format .......................................... 301
NLMNISGDw.d Format .......................................... 302
NLMNITHBw.d Format .......................................... 303
NLMNITRYw.d Format .......................................... 304
NLMNITWDw.d Format .......................................... 305
NLMNIUSDw.d Format .......................................... 306
NLMNIZARw.d Format .......................................... 307
NMLNLAE Dx.d Format .......................................... 308
NMLNLAUDw.d Format .......................................... 309
NMLNLBGNw.d Format .......................................... 310
NMLNLBRlw.d Format .......................................... 311
NMLNLCADw.d Format .......................................... 312
NMLNLCHFw.d Format .......................................... 313
NMLNLCHNw.d Format .......................................... 314
NMLNLCKd Format ............................................. 315
NMLNLDKw.d Format ............................................. 316
NMLNLEE Kw.d Format ............................................. 317
NMLNLEGp.d Format ............................................. 318
NMLNLEURw.d Format ............................................. 319
NMLNGBPw.d Format ............................................. 320
NMLNLHKDw.d Format ............................................. 321
NMLNLHRKw.d Format ............................................. 322
NMLNLHUFw.d Format ............................................. 323
NMLNLIDRw.d Format ............................................. 324
NMLNILLSw.d Format ............................................. 325
NMLNLIINRw.d Format ............................................. 326
NMLNJIPYw.d Format ............................................. 327
NMLNKRw.d Format ............................................. 328
NMLNLLTLw.d Format ............................................. 329
NMLNLVLw.d Format ............................................. 330
NMLNLOPw.d Format ............................................. 331
NMLNMXNw.d Format ............................................. 332
NMLNMRYw.d Format ............................................. 333
NMLNOKw.d Format ............................................. 334
NMLNLNZDw.d Format ............................................. 335
NMLNLPLNw.d Format ............................................. 336
NMLNLKRWw.d Format ............................................. 337
NMLNLSEKw.d Format ............................................. 338
Formats Documented in Other Publications

Additional formats are documented in these publications:

- “DS2 Formats” in *SAS DS2 Language Reference*
- “FedSQL Formats” in *SAS FedSQL Language Reference*

Formats by Category

There are five categories of formats in this list:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS</td>
<td>Instructs SAS to write character data values from character variables in CAS.</td>
</tr>
<tr>
<td>Character</td>
<td>Instructs SAS to write character data values from character variables.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>Instructs SAS to write data values from variables that represent dates, times, and datetimes.</td>
</tr>
</tbody>
</table>
Formats by Category

ISO 8601
Instructs SAS to write date, time, and datetime values using the ISO 8601 standard.

Numeric
Instructs SAS to write numeric data values from numeric variables.

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

Storing user-defined formats is an important consideration if you associate these formats with variables in permanent SAS data sets, especially those data sets shared with other users. For information about creating and storing user-defined formats, see “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.

<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS</td>
<td>$BASE64Xw. Format (p. 94)</td>
<td>Converts character data into ASCII text by using Base 64 encoding.</td>
</tr>
<tr>
<td></td>
<td>$BINARYw. Format (p. 95)</td>
<td>Converts character data to binary representation.</td>
</tr>
<tr>
<td></td>
<td>$CHARw. Format (p. 96)</td>
<td>Writes standard character data.</td>
</tr>
<tr>
<td></td>
<td>SCSTR Format (p. 97)</td>
<td>Looks for the last non-blank character of a character argument and passes a copy of the string with a null terminator after the last non-blank character.</td>
</tr>
<tr>
<td></td>
<td>$HEXw. Format (p. 100)</td>
<td>Converts character data to hexadecimal representation.</td>
</tr>
<tr>
<td></td>
<td>$OCTALw. Format (p. 116)</td>
<td>Converts character data to octal representation.</td>
</tr>
<tr>
<td></td>
<td>$QUOTEw. Format (p. 117)</td>
<td>Writes data values that are enclosed in double quotation marks.</td>
</tr>
<tr>
<td></td>
<td>SREVERJw. Format (p. 119)</td>
<td>Writes character data in reverse order and preserves blanks.</td>
</tr>
<tr>
<td></td>
<td>$REVERSw. Format (p. 120)</td>
<td>Writes character data in reverse order and left-aligns</td>
</tr>
<tr>
<td></td>
<td>$UPCASEw. Format (p. 121)</td>
<td>Converts character data to uppercase.</td>
</tr>
<tr>
<td></td>
<td>SUUIDw. Format (p. 122)</td>
<td>Converts character data to the Universally Unique Identifier (UUID) format.</td>
</tr>
<tr>
<td></td>
<td>BESTw. Format (p. 126)</td>
<td>SAS chooses the best notation.</td>
</tr>
<tr>
<td></td>
<td>BESTDw,p Format (p. 128)</td>
<td>Prints numeric values, lining up decimal places for values of similar magnitude, and prints integers without decimals.</td>
</tr>
<tr>
<td></td>
<td>COMMAw,d Format (p. 145)</td>
<td>Writes numeric values with a comma that separates every three digits and a period that separates the decimal fraction.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>COMMAXw.d Format</td>
<td>(p. 147)</td>
<td>Writes numeric values with a period that separates every three digits and a comma that separates the decimal fraction.</td>
</tr>
<tr>
<td>Dw.p Format</td>
<td>(p. 148)</td>
<td>Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude.</td>
</tr>
<tr>
<td>DATEw. Format</td>
<td>(p. 150)</td>
<td>Writes date values in the form <em>ddmmmyy</em>, <em>ddmmmyyyy</em>, or <em>dd-mm-yy</em>, either two or four digits.</td>
</tr>
<tr>
<td>DATEAMPMw.d Format</td>
<td>(p. 151)</td>
<td>Writes datetime values in the form <em>ddmmmyy:hh:mm:ss.ss</em> with AM or PM.</td>
</tr>
<tr>
<td>DATETIMEw.d Format</td>
<td>(p. 153)</td>
<td>Writes datetime values in the form <em>ddmmmyy:hh:mm:ss.ss</em>.</td>
</tr>
<tr>
<td>DAYw. Format</td>
<td>(p. 155)</td>
<td>Writes date values as the day of the month.</td>
</tr>
<tr>
<td>DDMMYYw. Format</td>
<td>(p. 156)</td>
<td>Writes date values in the form <em>ddmm&lt;yy&gt;</em> or <em>dd-mm-yy</em>, where a forward slash is the separator and the year appears as either two or four digits.</td>
</tr>
<tr>
<td>DDMMYYxw. Format</td>
<td>(p. 158)</td>
<td>Writes date values in the form <em>ddmm&lt;yy&gt;</em> or <em>dd-mm-yy&lt;yy&gt;</em> 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.</td>
</tr>
<tr>
<td>DOLLARw.d Format</td>
<td>(p. 160)</td>
<td>Writes numeric values with a leading dollar sign, a comma that separates every three digits, and a period that separates the decimal fraction.</td>
</tr>
<tr>
<td>DOLLARXw.d Format</td>
<td>(p. 161)</td>
<td>Writes numeric values with a leading dollar sign, a period that separates every three digits, and a comma that separates the decimal fraction.</td>
</tr>
<tr>
<td>DOWNAMEw. Format</td>
<td>(p. 163)</td>
<td>Writes date values as the name of the day of the week.</td>
</tr>
<tr>
<td>DTMONYYw. Format</td>
<td>(p. 165)</td>
<td>Writes the date part of a datetime value as the month and year in the format <em>mmmyy</em> or <em>mmmyyyy</em>.</td>
</tr>
<tr>
<td>DTWKDATXw. Format</td>
<td>(p. 166)</td>
<td>Writes the date part of a datetime value as the day of the week and the date in the form <em>day-of-week, dd month-name yy</em> (or <em>yyyy</em>).</td>
</tr>
<tr>
<td>DTYEARw. Format</td>
<td>(p. 168)</td>
<td>Writes the date part of a datetime value as the year in the format <em>yy</em> or <em>yyyy</em>.</td>
</tr>
<tr>
<td>DTYYQCw. Format</td>
<td>(p. 169)</td>
<td>Writes the date part of a datetime value as the year and the quarter and separates them with a colon (:).</td>
</tr>
<tr>
<td>Ew. Format</td>
<td>(p. 170)</td>
<td>Writes numeric values in scientific notation.</td>
</tr>
<tr>
<td>E8601DAw. Format</td>
<td>(p. 171)</td>
<td>Writes date values by using the ISO 8601 extended notation <em>yyyy-mm-dd</em>.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>E8601DNw. Format (p. 172)</td>
<td>Writes dates from SAS datetime values by using the ISO 8601 extended notation <code>yyyy-mm-dd</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601DTw.d Format (p. 173)</td>
<td>Writes datetime values by using the ISO 8601 extended notation <code>yyyy-mm-ddThh:mm:ss.ffffff</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601DXw. Format (p. 175)</td>
<td>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 <code>yyyy-mm-ddThh:mm:ss+hh:mm</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601DZw. Format (p. 176)</td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) by using the ISO 8601 datetime and time zone extended notation <code>yyyy-mm-ddThh:mm:ss+00:00</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601LXw. Format (p. 178)</td>
<td>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+</td>
<td>–hh:mm`.</td>
</tr>
<tr>
<td>E8601LZw. Format (p. 179)</td>
<td>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+</td>
<td>–hh:mm`.</td>
</tr>
<tr>
<td>E8601TMw.d Format (p. 181)</td>
<td>Writes time values by using the ISO 8601 extended notation <code>hh:mm:ss.ffffff</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601TXw. Format (p. 182)</td>
<td>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+</td>
<td>–hh:mm`.</td>
</tr>
<tr>
<td>E8601TZw.d Format (p. 184)</td>
<td>Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation `hh:mm:ss.&lt;fff&gt;+</td>
<td>–hh:mm`.</td>
</tr>
<tr>
<td>FLOATw.d Format (p. 191)</td>
<td>Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the <code>dth</code> power.</td>
<td></td>
</tr>
<tr>
<td>FRACTw. Format (p. 192)</td>
<td>Converts numeric values to fractions.</td>
<td></td>
</tr>
<tr>
<td>HEXw. Format (p. 193)</td>
<td>Converts real binary (floating-point) values to hexadecimal representation.</td>
<td></td>
</tr>
<tr>
<td>HHMMw.d Format (p. 195)</td>
<td>Writes time values as hours and minutes in the form <code>hh:mm</code>.</td>
<td></td>
</tr>
<tr>
<td>HOURw.d Format (p. 197)</td>
<td>Writes time values as hours and decimal fractions of hours.</td>
<td></td>
</tr>
<tr>
<td>IEEEw.d Format (p. 201)</td>
<td>Generates an IEEE floating-point value by multiplying a number by 10 raised to the <code>dth</code> power.</td>
<td></td>
</tr>
<tr>
<td>JULIANw. Format (p. 204)</td>
<td>Writes date values as Julian dates in the form <code>yyddd</code> or <code>yyyyyddd</code>.</td>
<td></td>
</tr>
<tr>
<td>MDYAMPMw.d Format (p. 205)</td>
<td>Writes datetime values in the form `mm/dd/yy&lt;yy&gt; hh:mm AM</td>
<td>PM`. The year can be either two or four digits.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MMDDYYw. Format</td>
<td></td>
<td>Writes date values in the form \textit{mmdd-yy or mm/dd-yy}, where a forward slash is the separator and the year appears as either two or four digits.</td>
</tr>
<tr>
<td>MMDDYYxw. Format</td>
<td></td>
<td>Writes date values in the form \textit{mmdd-yy or mm-dd-yy}, where the \textit{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.</td>
</tr>
<tr>
<td>MMSSw.d Format</td>
<td></td>
<td>Writes time values as the number of minutes and seconds since midnight.</td>
</tr>
<tr>
<td>MMYYw. Format</td>
<td></td>
<td>Writes date values in the form \textit{mmM-yy}, where M is the separator and the year appears as either two or four digits.</td>
</tr>
<tr>
<td>MMYYxw. Format</td>
<td></td>
<td>Writes date values in the form \textit{mm-yy or mm-yy}, where the \textit{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.</td>
</tr>
<tr>
<td>MONNAMEw. Format</td>
<td></td>
<td>Writes date values as the name of the month.</td>
</tr>
<tr>
<td>MONTHw. Format</td>
<td></td>
<td>Writes date values as the month of the year.</td>
</tr>
<tr>
<td>MONYYw. Format</td>
<td></td>
<td>Writes date values as the month and the year in the form \textit{mmmmyy or mmmmyyyy}.</td>
</tr>
<tr>
<td>NEGPARENw.d Format</td>
<td></td>
<td>Writes negative numeric values in parentheses.</td>
</tr>
<tr>
<td>NLBESTw. Format</td>
<td></td>
<td>Writes the best numerical notation based on the locale.</td>
</tr>
<tr>
<td>NLDATEw. Format</td>
<td></td>
<td>Converts a SAS date value to the date value of the specified locale, and then writes the date value as a date.</td>
</tr>
<tr>
<td>NLDATELw. Format</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form month, date, year.</td>
</tr>
<tr>
<td>NLDATEMw. Format</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in a medium-uniform pattern.</td>
</tr>
<tr>
<td>NLDATEMDw. Format</td>
<td></td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the value as the name of the month and the day of the month.</td>
</tr>
<tr>
<td>NLDATEMDLw. Format</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NLDATEMDw. Format (p. 229)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATEMDSw. Format (p. 230)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATEMNw. Format (p. 231)</td>
<td>Converts a SAS date value to the date value of the specified locale, and then writes the value as the name of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATESw. Format (p. 232)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date string.</td>
<td></td>
</tr>
<tr>
<td>NLDATEWw. Format (p. 233)</td>
<td>Converts a SAS date value to the date value of the specified locale, and then writes the value as the date and the day of the week.</td>
<td></td>
</tr>
<tr>
<td>NLDATEWNw. Format (p. 234)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the day of the week.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYMw. Format (p. 236)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the name of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYMLw. Format (p. 237)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the month and year.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYMMw. Format (p. 238)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date values as the month and year with abbreviations.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYMSw. Format (p. 239)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and year.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYQw. Format (p. 240)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the quarter.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYQLw. Format (p. 241)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year’s quarter value (Q1–Q4) using abbreviations.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYQMw. Format (p. 242)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year’s quarter value (Q1–Q4) using abbreviations.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYQSw. Format (p. 243)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year’s quarter value (1–4) with numbers and delimiters.</td>
<td></td>
</tr>
<tr>
<td>NLDATEYRw. Format (p. 244)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLDATEYWw. Format (p. 245)</td>
<td></td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the week.</td>
</tr>
<tr>
<td>NLDATMw. Format (p. 246)</td>
<td></td>
<td>Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime.</td>
</tr>
<tr>
<td>NLDATMAPw. Format (p. 247)</td>
<td></td>
<td>Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime with a.m. or p.m.</td>
</tr>
<tr>
<td>NLDATMDTw. Format (p. 248)</td>
<td></td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month, day of the month and year.</td>
</tr>
<tr>
<td>NLDATMLw. Format (p. 249)</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using abbreviations.</td>
</tr>
<tr>
<td>NLDATMMw. Format (p. 250)</td>
<td></td>
<td>Converts a SAS datetime value to the datetime string of the specified locale in the medium representation of the date.</td>
</tr>
<tr>
<td>NLDATMMDw. Format (p. 251)</td>
<td></td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month and the day of the month.</td>
</tr>
<tr>
<td>NLDATMMDLw. Format (p. 252)</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the full-length of the month and day of the month.</td>
</tr>
<tr>
<td>NLDATMMDMw. Format (p. 253)</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using numbers and delimiters.</td>
</tr>
<tr>
<td>NLDATMMDSw. Format (p. 254)</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using numbers and delimiters.</td>
</tr>
<tr>
<td>NLDATMMNw. Format (p. 255)</td>
<td></td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month.</td>
</tr>
<tr>
<td>NLDATMSw. Format (p. 255)</td>
<td></td>
<td>Converts a SAS datetime value to the datetime string of the specified locale in the short representation of the date.</td>
</tr>
<tr>
<td>NLDATMTw. Format (p. 256)</td>
<td></td>
<td>Converts the time portion of a SAS datetime value to the time-of-day value of the specified locale, and then writes the value as a time of day.</td>
</tr>
<tr>
<td>NLDATMTZw. Format (p. 258)</td>
<td></td>
<td>Converts the time portion of the SAS datetime value to the time of day and time zone of the specified locale.</td>
</tr>
<tr>
<td>NLDATMWw. Format (p. 258)</td>
<td></td>
<td>Converts SAS datetime values to the locale sensitive datetime string as the day of the week and the datetime.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLDATMWNw. Format (p. 260)</td>
<td>Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as the day of the week.</td>
<td></td>
</tr>
<tr>
<td>NLDATMZWw. Format (p. 260)</td>
<td>Converts SAS date values of the specified locale to a day-of-week, datetime, and time zone value.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYMw. Format (p. 261)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the month and year.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYMLw. Format (p. 262)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYMMw. Format (p. 263)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYMSw. Format (p. 264)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the month and year with numbers and a delimiter.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYQMw. Format (p. 265)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the quarter and the year.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYQMMw. Format (p. 267)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year’s quarter (1–4) and the year.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYQSw. Format (p. 268)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the quarter (1–4) using numbers and a delimiter.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYRWw. Format (p. 269)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year.</td>
<td></td>
</tr>
<tr>
<td>NLDAMYWw. Format (p. 270)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the week number and the year.</td>
<td></td>
</tr>
<tr>
<td>NLDATMZWw. Format (p. 270)</td>
<td>Converts SAS datetime values to the locale-sensitive datetime string as datetime and time zone.</td>
<td></td>
</tr>
<tr>
<td>NLMNIAEDw.d Format (p. 271)</td>
<td>Writes the monetary format of the international expression for the United Arab Emirates.</td>
<td></td>
</tr>
<tr>
<td>NLMNIAUDw.d Format (p. 272)</td>
<td>Writes the monetary format of the international expression for Australia.</td>
<td></td>
</tr>
<tr>
<td>NLMNIBGNgw.d Format (p. 273)</td>
<td>Writes the monetary format of the international expression for Bulgaria.</td>
<td></td>
</tr>
<tr>
<td>NLMNIBRLw.d Format (p. 274)</td>
<td>Writes the monetary format of the international expression for Brazil.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NLMNICADw.d Format (p. 275)</td>
<td>Writes the monetary format of the international expression for Canada.</td>
<td></td>
</tr>
<tr>
<td>NLMNICHFw.d Format (p. 276)</td>
<td>Writes the monetary format of the international expression for Liechtenstein and Switzerland.</td>
<td></td>
</tr>
<tr>
<td>NLMNICNYw.d Format (p. 277)</td>
<td>Writes the monetary format of the international expression for China.</td>
<td></td>
</tr>
<tr>
<td>NLMNICZKw.d Format (p. 278)</td>
<td>Writes the monetary format of the international expression for the Czech Republic.</td>
<td></td>
</tr>
<tr>
<td>NLMNIDKKw.d Format (p. 279)</td>
<td>Writes the monetary format of the international expression for Denmark, Faroe Island, and Greenland.</td>
<td></td>
</tr>
<tr>
<td>NLMNIEEKw.d Format (p. 280)</td>
<td>Writes the monetary format of the international expression for Estonia.</td>
<td></td>
</tr>
<tr>
<td>NLMNIEGPw.d Format (p. 281)</td>
<td>Writes the monetary format of the international expression for Egypt.</td>
<td></td>
</tr>
<tr>
<td>NLMNIEURw.d Format (p. 282)</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>NLMNIGBPw.d Format (p. 283)</td>
<td>Writes the monetary format of the international expression for the United Kingdom.</td>
<td></td>
</tr>
<tr>
<td>NLMNIHKDw.d Format (p. 284)</td>
<td>Writes the monetary format of the international expression for Hong Kong.</td>
<td></td>
</tr>
<tr>
<td>NLMNIHRKw.d Format (p. 285)</td>
<td>Writes the monetary format of the international expression for Croatia.</td>
<td></td>
</tr>
<tr>
<td>NLMNIHUFw.d Format (p. 286)</td>
<td>Writes the monetary format of the international expression for Hungary.</td>
<td></td>
</tr>
<tr>
<td>NLMNIIDRw.d Format (p. 287)</td>
<td>Writes the monetary format of the international expression for Indonesia.</td>
<td></td>
</tr>
<tr>
<td>NLMNIIILSw.d Format (p. 288)</td>
<td>Writes the monetary format of the international expression for Israel.</td>
<td></td>
</tr>
<tr>
<td>NLMNIIINRw.d Format (p. 289)</td>
<td>Writes the monetary format of the international expression for India.</td>
<td></td>
</tr>
<tr>
<td>NLMNIKRWw.d Format (p. 291)</td>
<td>Writes the monetary format of the international expression for South Korea.</td>
<td></td>
</tr>
<tr>
<td>NLMNILTLw.d Format (p. 292)</td>
<td>Writes the monetary format of the international expression for Lithuania.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NLMNIVL</td>
<td>\texttt{w.d Format (p. 293)}</td>
<td>Writes the monetary format of the international expression for Latvia.</td>
</tr>
<tr>
<td>NLMNIMOP</td>
<td>\texttt{w.d Format (p. 294)}</td>
<td>Writes the monetary format of the international expression for Macau.</td>
</tr>
<tr>
<td>NLMNIMXN</td>
<td>\texttt{w.d Format (p. 295)}</td>
<td>Writes the monetary format of the international expression for Mexico.</td>
</tr>
<tr>
<td>NLMNIMYR</td>
<td>\texttt{w.d Format (p. 296)}</td>
<td>Writes the monetary format of the international expression for Malaysia.</td>
</tr>
<tr>
<td>NLMNINOK</td>
<td>\texttt{w.d Format (p. 297)}</td>
<td>Writes the monetary format of the international expression for Norway.</td>
</tr>
<tr>
<td>NLMNINZD</td>
<td>\texttt{w.d Format (p. 298)}</td>
<td>Writes the monetary format of the international expression for New Zealand.</td>
</tr>
<tr>
<td>NLMNIPLN</td>
<td>\texttt{w.d Format (p. 299)}</td>
<td>Writes the monetary format of the international expression for Poland.</td>
</tr>
<tr>
<td>NLMNIRUB</td>
<td>\texttt{w.d Format (p. 300)}</td>
<td>Writes the monetary format of the international expression for Russia.</td>
</tr>
<tr>
<td>NLMNISEK</td>
<td>\texttt{w.d Format (p. 301)}</td>
<td>Writes the monetary format of the international expression for Sweden.</td>
</tr>
<tr>
<td>NLMNISGD</td>
<td>\texttt{w.d Format (p. 302)}</td>
<td>Writes the monetary format of the international expression for Singapore.</td>
</tr>
<tr>
<td>NLMNIUTHB</td>
<td>\texttt{w.d Format (p. 303)}</td>
<td>Writes the monetary format of the international expression for Thailand.</td>
</tr>
<tr>
<td>NLMNITRY</td>
<td>\texttt{w.d Format (p. 304)}</td>
<td>Writes the monetary format of the international expression for Turkey.</td>
</tr>
<tr>
<td>NLMNITWD</td>
<td>\texttt{w.d Format (p. 305)}</td>
<td>Writes the monetary format of the international expression for Taiwan.</td>
</tr>
<tr>
<td>NLMNIUSD</td>
<td>\texttt{w.d Format (p. 306)}</td>
<td>Writes the monetary format of the international expression for Puerto Rico and the United States.</td>
</tr>
<tr>
<td>NLMNIZAR</td>
<td>\texttt{w.d Format (p. 307)}</td>
<td>Writes the monetary format of the international expression for South Africa.</td>
</tr>
<tr>
<td>NLMNLAED</td>
<td>\texttt{x.d Format (p. 308)}</td>
<td>Writes the monetary format of the local expression for the United Arab Emirates.</td>
</tr>
<tr>
<td>NLMNLAUD</td>
<td>\texttt{w.d Format (p. 309)}</td>
<td>Writes the monetary format of the local expression for Australia.</td>
</tr>
<tr>
<td>NLMNLBGN</td>
<td>\texttt{w.d Format (p. 310)}</td>
<td>Writes the monetary format of the local expression for Bulgaria.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLMNLBRLw.d Format</td>
<td>(p. 311)</td>
<td>Writes the monetary format of the local expression for Brazil.</td>
</tr>
<tr>
<td>NLMNCADw.d Format</td>
<td>(p. 312)</td>
<td>Writes the monetary format of the local expression for Canada.</td>
</tr>
<tr>
<td>NLMNCHFW.d Format</td>
<td>(p. 313)</td>
<td>Writes the monetary format of the local expression for Liechtenstein and Switzerland.</td>
</tr>
<tr>
<td>NLMNLCNYw.d Format</td>
<td>(p. 314)</td>
<td>Writes the monetary format of the local expression for China.</td>
</tr>
<tr>
<td>NLMNLCZKw.d Format</td>
<td>(p. 315)</td>
<td>Writes the monetary format of the local expression for the Czech Republic.</td>
</tr>
<tr>
<td>NLMNLDKKw.d Format</td>
<td>(p. 316)</td>
<td>Writes the monetary format of the local expression for Denmark, Faroe Island, and Greenland.</td>
</tr>
<tr>
<td>NLMNLEEKw.d Format</td>
<td>(p. 317)</td>
<td>Writes the monetary format of the local expression for Estonia.</td>
</tr>
<tr>
<td>NLMNLEGpw.d Format</td>
<td>(p. 318)</td>
<td>Writes the monetary format of the local expression for Egypt.</td>
</tr>
<tr>
<td>NLMNLEURw.d Format</td>
<td>(p. 319)</td>
<td>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.</td>
</tr>
<tr>
<td>NLMNLGBPw.d Format</td>
<td>(p. 320)</td>
<td>Writes the monetary format of the local expression for the United Kingdom.</td>
</tr>
<tr>
<td>NLMNHLHKDw.d Format</td>
<td>(p. 321)</td>
<td>Writes the monetary format of the local expression for Hong Kong.</td>
</tr>
<tr>
<td>NLMNHLRKw.d Format</td>
<td>(p. 322)</td>
<td>Writes the monetary format of the local expression for Croatia.</td>
</tr>
<tr>
<td>NLMNHLHUFw.d Format</td>
<td>(p. 323)</td>
<td>Writes the monetary format of the local expression for Hungary.</td>
</tr>
<tr>
<td>NLMNLDIRw.d Format</td>
<td>(p. 324)</td>
<td>Writes the monetary format of the local expression for Indonesia.</td>
</tr>
<tr>
<td>NLMNLILSw.d Format</td>
<td>(p. 325)</td>
<td>Writes the monetary format of the local expression for Israel.</td>
</tr>
<tr>
<td>NLMNLINRw.d Format</td>
<td>(p. 326)</td>
<td>Writes the monetary format of the local expression for India.</td>
</tr>
<tr>
<td>NLMNJPYw.d Format</td>
<td>(p. 327)</td>
<td>Writes the monetary format of the international expression for Japan.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NLMNLKRWw.d Format (p. 328)</td>
<td>Writes the monetary format of the local expression for South Korea.</td>
<td></td>
</tr>
<tr>
<td>NLMNLLTLw.d Format (p. 329)</td>
<td>Writes the monetary format of the local expression for Lithuania.</td>
<td></td>
</tr>
<tr>
<td>NLMNLLVLw.d Format (p. 330)</td>
<td>Writes the monetary format of the local expression for Latvia.</td>
<td></td>
</tr>
<tr>
<td>NLMNLMOPw.d Format (p. 331)</td>
<td>Writes the monetary format of the local expression for Macau.</td>
<td></td>
</tr>
<tr>
<td>NLMNLMXNw.d Format (p. 332)</td>
<td>Writes the monetary format of the local expression for Mexico.</td>
<td></td>
</tr>
<tr>
<td>NLMNLMYRw.d Format (p. 333)</td>
<td>Writes the monetary format of the local expression for Malaysia.</td>
<td></td>
</tr>
<tr>
<td>NLMNLNOKw.d Format (p. 334)</td>
<td>Writes the monetary format of the local expression for Norway.</td>
<td></td>
</tr>
<tr>
<td>NLMNZNZWw.d Format (p. 335)</td>
<td>Writes the monetary format of the local expression for New Zealand.</td>
<td></td>
</tr>
<tr>
<td>NLMNPLNWw.d Format (p. 336)</td>
<td>Writes the monetary format of the local expression for Poland.</td>
<td></td>
</tr>
<tr>
<td>NLMNLRUBw.d Format (p. 337)</td>
<td>Writes the monetary format of the local expression for Russia.</td>
<td></td>
</tr>
<tr>
<td>NLMNISEKw.d Format (p. 338)</td>
<td>Writes the monetary format of the local expression for Sweden.</td>
<td></td>
</tr>
<tr>
<td>NLMNLSGDw.d Format (p. 339)</td>
<td>Writes the monetary format of the local expression for Singapore.</td>
<td></td>
</tr>
<tr>
<td>NLMNLTHBWw.d Format (p. 340)</td>
<td>Writes the monetary format of the local expression for Thailand.</td>
<td></td>
</tr>
<tr>
<td>NLMNLTRYw.d Format (p. 341)</td>
<td>Writes the monetary format of the local expression for Turkey.</td>
<td></td>
</tr>
<tr>
<td>NLMNLTWDw.d Format (p. 342)</td>
<td>Writes the monetary format of the local expression for Taiwan.</td>
<td></td>
</tr>
<tr>
<td>NLMNLUSDw.d Format (p. 343)</td>
<td>Writes the monetary format of the local expression for Puerto Rico and the United States.</td>
<td></td>
</tr>
<tr>
<td>NLMNLZARw.d Format (p. 344)</td>
<td>Writes the monetary format of the local expression for South Africa.</td>
<td></td>
</tr>
<tr>
<td>NLMNYw.d Format (p. 345)</td>
<td>Writes the monetary format of the local expression in the specified locale using local currency.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>NLMNYw.d Format (p. 346)</td>
<td>Writes the monetary format of the international expression in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLNUMw.d Format (p. 347)</td>
<td>Writes the numeric format of the local expression in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLNUMIw.d Format (p. 349)</td>
<td>Writes the numeric format of the international expression in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLPCTw.d Format (p. 350)</td>
<td>Writes percentage data of the local expression in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLPCTIw.d Format (p. 352)</td>
<td>Writes percentage data of the international expression in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLPCNTw.d Format (p. 353)</td>
<td>Produces percentages, using a minus sign for negative values.</td>
</tr>
<tr>
<td></td>
<td>NLPCTPw.d Format (p. 354)</td>
<td>Writes locale-specific numeric values as percentages.</td>
</tr>
<tr>
<td></td>
<td>NLPVALUEw.d Format (p. 355)</td>
<td>Writes p-values of the local expression in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLSTRMONw.d Format (p. 356)</td>
<td>Writes the month name in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLSTRQTRw.d Format (p. 357)</td>
<td>Writes a numeric value as the quarter-of-the-year in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLSTRWKw.d Format (p. 359)</td>
<td>Writes a numeric value as the day-of-the-week in the specified locale.</td>
</tr>
<tr>
<td></td>
<td>NLTIMAPw. Format (p. 360)</td>
<td>Converts a SAS time value to the time value of a specified locale, and then writes the value as a time value with a.m. or p.m. NLTIMAP also converts SAS date-time values.</td>
</tr>
<tr>
<td></td>
<td>NLTIMEw. Format (p. 361)</td>
<td>Converts a SAS time value to the time value of the specified locale, and then writes the value as a time value. NLTIME also converts SAS date-time values.</td>
</tr>
<tr>
<td></td>
<td>OCTALw. Format (p. 364)</td>
<td>Converts numeric values to octal representation.</td>
</tr>
<tr>
<td></td>
<td>PERCENTw.d Format (p. 371)</td>
<td>Writes numeric values as percentages.</td>
</tr>
<tr>
<td></td>
<td>PERCENTNw.d Format (p. 372)</td>
<td>Produces percentages, using a minus sign for negative values.</td>
</tr>
<tr>
<td></td>
<td>QTRw. Format (p. 380)</td>
<td>Writes date values as the quarter of the year.</td>
</tr>
<tr>
<td></td>
<td>QTRRw. Format (p. 381)</td>
<td>Writes date values as the quarter of the year in Roman numerals.</td>
</tr>
<tr>
<td></td>
<td>ROMANw. Format (p. 383)</td>
<td>Writes numeric values as Roman numerals.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>TIMEw.d Format (p. 402)</td>
<td></td>
<td>Writes time values as hours, minutes, and seconds in the form hh:mm:ss.ss.</td>
</tr>
<tr>
<td>TIMEAMPMw.d Format (p. 405)</td>
<td></td>
<td>Writes time and datetime values as hours, minutes, and seconds in the form hh:mm:ss.ss with AM or PM.</td>
</tr>
<tr>
<td>TODw.d Format (p. 406)</td>
<td></td>
<td>Writes SAS time values and the time portion of SAS datetime values in the form hh:mm:ss.ss.</td>
</tr>
<tr>
<td>VAXRBw.d Format (p. 409)</td>
<td></td>
<td>Writes real binary (floating-point) data in VMS format.</td>
</tr>
<tr>
<td>w.d Format (p. 411)</td>
<td></td>
<td>Writes standard numeric data one digit per byte.</td>
</tr>
<tr>
<td>WEEKDATXw. Format (p. 414)</td>
<td></td>
<td>Writes date values as the day of the week and date in the form day-of-week, dd month-name yy (or yyyy).</td>
</tr>
<tr>
<td>WEEKDAYw. Format (p. 416)</td>
<td></td>
<td>Writes date values as the day of the week.</td>
</tr>
<tr>
<td>YEARw. Format (p. 427)</td>
<td></td>
<td>Writes date values as the year.</td>
</tr>
<tr>
<td>YYMMw. Format (p. 429)</td>
<td></td>
<td>Writes date values in the form &lt;yy&gt;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.</td>
</tr>
<tr>
<td>YYMMDDw. Format (p. 430)</td>
<td></td>
<td>Writes date values in the form yymmd or &lt;yy&gt;yy-mm-dd, where a hyphen is the separator and the year appears as either two or four digits.</td>
</tr>
<tr>
<td>YYMMDDxw. Format (p. 432)</td>
<td></td>
<td>Writes date values in the form yymmd or &lt;yy&gt;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.</td>
</tr>
<tr>
<td>YYMMxw. Format (p. 434)</td>
<td></td>
<td>Writes date values in the form &lt;yy&gt;yymm or &lt;yy&gt;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.</td>
</tr>
<tr>
<td>YYMONw. Format (p. 436)</td>
<td></td>
<td>Writes date values in the form yymmm or yyyyymmm.</td>
</tr>
<tr>
<td>YYQw. Format (p. 438)</td>
<td></td>
<td>Writes date values in the form &lt;yy&gt;yyQq, where Q is the separator, the year appears as either two or four digits, and q is the quarter of the year.</td>
</tr>
<tr>
<td>YYQxw. Format (p. 439)</td>
<td></td>
<td>Writes date values in the form &lt;yy&gt;yyq or &lt;yy&gt;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.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>YYQRw. Format (p. 441)</td>
<td>Writes date values in the form <code>&lt;yy&gt;yyQqr</code>, 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.</td>
<td></td>
</tr>
<tr>
<td>YYQRxw. Format (p. 442)</td>
<td>Writes date values in the form <code>&lt;yy&gt;yyqr</code> or <code>&lt;yy&gt;yy-qr</code>, 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.</td>
<td></td>
</tr>
<tr>
<td>YYQZw. Format (p. 444)</td>
<td>Writes SAS date values in the form <code>&lt;yy&gt;&lt;qq&gt;</code>, the year appears as 2 or 4 digits, and qq is the quarter of the year.</td>
<td></td>
</tr>
<tr>
<td>YYWEEKUw. Format (p. 446)</td>
<td>Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.</td>
<td></td>
</tr>
<tr>
<td>YYWEEKVw. Format (p. 447)</td>
<td>Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.</td>
<td></td>
</tr>
<tr>
<td>YYWEEKWw. Format (p. 449)</td>
<td>Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.</td>
<td></td>
</tr>
<tr>
<td>Zw.d Format (p. 450)</td>
<td>Writes standard numeric data with leading 0s.</td>
<td></td>
</tr>
<tr>
<td>Character</td>
<td>$ASCIIw. Format (p. 93)</td>
<td>Converts native format character data to ASCII representation.</td>
</tr>
<tr>
<td></td>
<td>$BASE64Xw. Format (p. 94)</td>
<td>Converts character data into ASCII text by using Base 64 encoding.</td>
</tr>
<tr>
<td></td>
<td>$BINARYw. Format (p. 95)</td>
<td>Converts character data to binary representation.</td>
</tr>
<tr>
<td></td>
<td>$CHARw. Format (p. 96)</td>
<td>Writes standard character data.</td>
</tr>
<tr>
<td></td>
<td>$CSTR Format (p. 97)</td>
<td>Looks for the last non-blank character of a character argument and passes a copy of the string with a null terminator after the last non-blank character.</td>
</tr>
<tr>
<td></td>
<td>$EBCDICw. Format (p. 98)</td>
<td>Converts native format character data to EBCDIC representation.</td>
</tr>
<tr>
<td></td>
<td>$HEXw. Format (p. 100)</td>
<td>Converts character data to hexadecimal representation.</td>
</tr>
<tr>
<td></td>
<td>$MSGCASEw. Format (p. 101)</td>
<td>Writes character data in uppercase when the MSGCASE system option is in effect.</td>
</tr>
<tr>
<td></td>
<td>$OCTALw. Format (p. 116)</td>
<td>Converts character data to octal representation.</td>
</tr>
<tr>
<td></td>
<td>$REVERJw. Format (p. 117)</td>
<td>Writes character data in reverse order and preserves blanks.</td>
</tr>
<tr>
<td></td>
<td>$REVERSew. Format (p. 120)</td>
<td>Writes character data in reverse order and left-aligns</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$\text{SUPCASEw}$. Format (p. 121)</td>
<td>Converts character data to uppercase.</td>
<td></td>
</tr>
<tr>
<td>$\text{UUIDw}$. Format (p. 122)</td>
<td>Converts character data to the Universally Unique Identifier (UUID) format.</td>
<td></td>
</tr>
<tr>
<td>$\text{VARYINGw}$. Format (p. 123)</td>
<td>Writes character data of varying length.</td>
<td></td>
</tr>
<tr>
<td>$\text{sw}$. Format (p. 125)</td>
<td>Writes standard character data.</td>
<td></td>
</tr>
<tr>
<td>Date and Time</td>
<td>$\text{SN8601Bw.d Format (p. 102)}$</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations $\text{PnYnMnDTnHnMnS}$ and $\text{yyyy-mm-ddT hh:mm:ss}$.</td>
</tr>
<tr>
<td></td>
<td>$\text{SN8601BAw.d Format (p. 104)}$</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations $\text{Pyyyy-mm-ddT hh:mm:ss}$ and $\text{yyyy-mm-ddT hh:mm:ss}$.</td>
</tr>
<tr>
<td></td>
<td>$\text{SN8601Ew.d Format (p. 106)}$</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations $\text{PnYnMnDTnHnMnS}$ and $\text{yyyy-mm-ddT hh:mm:ss}$.</td>
</tr>
<tr>
<td></td>
<td>$\text{SN8601EAw.d Format (p. 107)}$</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations $\text{Pyyyy-mm-ddT hh:mm:ss}$ and $\text{yyyy-mm-ddT hh:mm:ss}$.</td>
</tr>
<tr>
<td></td>
<td>$\text{SN8601EHw.d Format (p. 109)}$</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations $\text{Pyyyy-mm-ddT hh:mm:ss}$ and $\text{yyyy-mm-ddT hh:mm:ss}$, using a hyphen (-) for omitted components.</td>
</tr>
<tr>
<td></td>
<td>$\text{SN8601EXw.d Format (p. 111)}$</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations $\text{Pyyyy-mm-ddT hh:mm:ss}$ and $\text{yyyy-mm-ddT hh:mm:ss}$, using an x for each digit of an omitted component.</td>
</tr>
<tr>
<td></td>
<td>$\text{SN8601Hw.d Format (p. 113)}$</td>
<td>Writes ISO 8601 duration, datetime, and interval forms $\text{PnYnMnDTnHnMnS}$ and $\text{yyyy-mm-ddT hh:mm:ss}$, dropping omitted components in duration values and using a hyphen (-) for omitted components in datetime values.</td>
</tr>
<tr>
<td></td>
<td>$\text{SN8601Xw.d Format (p. 115)}$</td>
<td>Writes ISO 8601 duration, datetime, and interval forms $\text{PnYnMnDTnHnMnS}$ and $\text{yyyy-mm-ddT hh:mm:ss}$, dropping omitted components in duration values and using an x for each digit of an omitted component in datetime values.</td>
</tr>
<tr>
<td></td>
<td>$\text{B8601DAw. Format (p. 131)}$</td>
<td>Writes date values by using the ISO 8601 basic notation $\text{yyyy-mmdd}$.</td>
</tr>
<tr>
<td></td>
<td>$\text{B8601DNw. Format (p. 132)}$</td>
<td>Writes dates from datetime values by using the ISO 8601 basic notation $\text{yyyy-mmdd}$.</td>
</tr>
<tr>
<td></td>
<td>$\text{B8601DTw.d Format (p. 133)}$</td>
<td>Writes datetime values by using the ISO 8601 basic notation $\text{yyyy-mmddT hh:mm:ss&lt;ffffff&gt;}$</td>
</tr>
<tr>
<td></td>
<td>$\text{B8601DXw. Format (p. 135)}$</td>
<td>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 $\text{yyyy-mmddT hh:mm+hhmm}$.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>B8601DZW Format (p. 136)</td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone basic notation <code>yyyyymmddThhmmss+0000</code>.</td>
<td></td>
</tr>
<tr>
<td>B8601LXW Format (p. 138)</td>
<td>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 <code>yyyyymmddThhmmss±hhmm</code>.</td>
<td></td>
</tr>
<tr>
<td>B8601LZW Format (p. 139)</td>
<td>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 <code>hhmmss±hhmm</code>.</td>
<td></td>
</tr>
</tbody>
</table>
| B8601TMw.d Format (p. 141) | Writes time values by using the ISO 8601 basic notation `hhmmss<fff>`.
<p>| B8601TXW Format (p. 142) | 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 <code>hhmmss±hhmm</code>. |
| B8601TZW Format (p. 143) | Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 basic time notation <code>hhmmss±hhmm</code>. |
| DATEw. Format (p. 150) | Writes date values in the form <code>ddmmmyy</code>, <code>ddmmyyyy</code>, or <code>dd-mm-yy&lt;yy&gt;</code>, where a forward slash is the separator and the year appears as either two or four digits. |
| DATEAMPMw.d Format (p. 151) | Writes datetime values in the form <code>ddmmyy:hh:mm:ss.ss</code> with AM or PM. |
| DATETIMEw.d Format (p. 153) | Writes datetime values in the form <code>ddmmyy:hh:mm:ss.ss</code>. |
| DAYw. Format (p. 155) | Writes date values as the day of the month. |
| DDMMYYw. Format (p. 156) | Writes date values in the form <code>ddmm&lt;yy&gt;yy</code> or <code>dd/mm/&lt;yy&gt;yy</code>, where a forward slash is the separator and the year appears as either two or four digits. |
| DDMMYYxw. Format (p. 158) | Writes date values in the form <code>ddmm&lt;yy&gt;yy</code> or <code>dd-mm-yy&lt;yy&gt;</code>, 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. 163) | Writes date values as the name of the day of the week. |
| DTDATEm. Format (p. 164) | Expects a datetime value as input and writes date values in the form <code>ddmmyy</code> or <code>ddmmyyyy</code>. |
| DTMONYYw. Format (p. 165) | Writes the date part of a datetime value as the month and year in the form <code>mmmyy</code> or <code>mmmyyyy</code>. |
| DTWKGDATXw. Format (p. 166) | Writes the date part of a datetime value as the day of the week and the date in the form <code>day-of-week, dd month-name yy</code> (or <code>yyyy</code>). |</p>
<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTYEARw</td>
<td>Format (p. 168)</td>
<td>Writes the date part of a datetime value as the year in the form yy or yyyy.</td>
</tr>
<tr>
<td>DTYYYQCw</td>
<td>Format (p. 169)</td>
<td>Writes the date part of a datetime value as the year and the quarter and separates them with a colon (:).</td>
</tr>
<tr>
<td>E8601DAw</td>
<td>Format (p. 171)</td>
<td>Writes date values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DNw</td>
<td>Format (p. 172)</td>
<td>Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DTw.d</td>
<td>Format (p. 173)</td>
<td>Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddTh:mm:ss.</td>
</tr>
<tr>
<td>E8601DXw</td>
<td>Format (p. 175)</td>
<td>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-ddTh:mm:ss+hh:mm.</td>
</tr>
<tr>
<td>E8601DZw</td>
<td>Format (p. 176)</td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddTh:mm:ss+00:00.</td>
</tr>
<tr>
<td>E8601LXw</td>
<td>Format (p. 178)</td>
<td>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-ddTh:mm:ss+</td>
</tr>
<tr>
<td>E8601LZw</td>
<td>Format (p. 179)</td>
<td>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+</td>
</tr>
<tr>
<td>E8601TMw.d</td>
<td>Format (p. 181)</td>
<td>Writes time values by using the ISO 8601 extended notation hh:mm:ss.</td>
</tr>
<tr>
<td>E8601TXw</td>
<td>Format (p. 182)</td>
<td>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+</td>
</tr>
<tr>
<td>E8601TZw.d</td>
<td>Format (p. 184)</td>
<td>Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation hh:mm:ss.&lt;fff&gt;+</td>
</tr>
<tr>
<td>HHMMw.d</td>
<td>Format (p. 195)</td>
<td>Writes time values as hours and minutes in the form hh:mm.</td>
</tr>
<tr>
<td>HOURw.d</td>
<td>Format (p. 197)</td>
<td>Writes time values as hours and decimal fractions of hours.</td>
</tr>
<tr>
<td>JULDAYw</td>
<td>Format (p. 202)</td>
<td>Writes date values as the Julian day of the year.</td>
</tr>
<tr>
<td>JULIANw</td>
<td>Format (p. 204)</td>
<td>Writes date values as Julian dates in the form yyddd or yyyyymdd.</td>
</tr>
<tr>
<td>MDYAMPMw.d</td>
<td>Format (p. 205)</td>
<td>Writes datetime values in the form mm/dd/yy&lt;yy&gt; hh:mm AM</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>MMDDYYw. Format (p. 207)</td>
<td></td>
<td>Writes date values in the form <em>mmdd&lt;yy&gt;yy</em> or <em>mm/dd&lt;yy&gt;yy</em>, where a forward slash is the separator and the year appears as either two or four digits.</td>
</tr>
<tr>
<td>MMDDYYxw. Format (p. 209)</td>
<td></td>
<td>Writes date values in the form <em>mmdd&lt;yy&gt;yy</em> or <em>mm-dd&lt;yy&gt;yy</em>, where the <em>x</em> 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.</td>
</tr>
<tr>
<td>MMSSw.d Format (p. 211)</td>
<td></td>
<td>Writes time values as the number of minutes and seconds since midnight.</td>
</tr>
<tr>
<td>MMYyw. Format (p. 212)</td>
<td></td>
<td>Writes date values in the form <em>mmM&lt;yy&gt;yy</em>, where <em>M</em> is the separator and the year appears as either two or four digits.</td>
</tr>
<tr>
<td>MMYyw. Format (p. 214)</td>
<td></td>
<td>Writes date values in the form <em>mm&lt;yy&gt;yy</em> or <em>mm-mm&lt;yy&gt;yy</em>, where the <em>x</em> 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.</td>
</tr>
<tr>
<td>MONNAMEw. Format (p. 216)</td>
<td></td>
<td>Writes date values as the name of the month.</td>
</tr>
<tr>
<td>MONTHw. Format (p. 217)</td>
<td></td>
<td>Writes date values as the month of the year.</td>
</tr>
<tr>
<td>MONYYw. Format (p. 218)</td>
<td></td>
<td>Writes date values as the month and the year in the form <em>mmm&lt;yy&gt;yy</em> or <em>mmm&lt;yy&gt;yyyy</em>.</td>
</tr>
<tr>
<td>NENGOw. Format (p. 221)</td>
<td></td>
<td>Writes date values as Japanese dates in the form <em>e.yymmdd</em>.</td>
</tr>
<tr>
<td>NLDATewan. Format (p. 224)</td>
<td></td>
<td>Converts a SAS date value to the date value of the specified locale, and then writes the date value as a date.</td>
</tr>
<tr>
<td>NLDATELwan. Format (p. 225)</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form month, date, year.</td>
</tr>
<tr>
<td>NLDATEMwan. Format (p. 226)</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in a medium-uniform pattern.</td>
</tr>
<tr>
<td>NLDATEMDwan. Format (p. 227)</td>
<td></td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the value as the name of the month and the day of the month.</td>
</tr>
<tr>
<td>NLDATEMDLwan. Format (p. 228)</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.</td>
</tr>
<tr>
<td>NLDATEMDMwan. Format (p. 229)</td>
<td></td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NLDATEMDSw. Format (p. 230)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATEMNNw. Format (p. 231)</td>
<td>Converts a SAS date value to the date value of the specified locale, and then writes the value as the name of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATESw. Format (p. 232)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date string.</td>
<td></td>
</tr>
<tr>
<td>NLDATENww. Format (p. 233)</td>
<td>Converts a SAS date value to the date value of the specified locale, and then writes the value as the date and the day of the week.</td>
<td></td>
</tr>
<tr>
<td>NLDATENWWn. Format (p. 234)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the day of the week.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYMw. Format (p. 236)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the name of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYMLw. Format (p. 237)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the month and year.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYMMww. Format (p. 238)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date values as the month and year with abbreviations.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYMSww. Format (p. 239)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and year.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYQww. Format (p. 240)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the quarter.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYQLww. Format (p. 241)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year’s quarter value (Q1–Q4) using abbreviations.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYQMww. Format (p. 242)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year’s quarter value (Q1–Q4) using abbreviations.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYQSw. Format (p. 243)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year’s quarter value (1–4) with numbers and delimiters.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYRWww. Format (p. 244)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year.</td>
<td></td>
</tr>
<tr>
<td>NLDATENYRWww. Format (p. 245)</td>
<td>Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the week.</td>
<td></td>
</tr>
<tr>
<td>NLDATENMww. Format (p. 246)</td>
<td>Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLDATMAPw. Format (p. 247)</td>
<td>Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime with a.m. or p.m.</td>
<td></td>
</tr>
<tr>
<td>NLDATMDTW. Format (p. 248)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month, day of the month and year.</td>
<td></td>
</tr>
<tr>
<td>NLDATMLw. Format (p. 249)</td>
<td>Converts a SAS datetime value to the datetime string of the specified locale in the long representation of the date.</td>
<td></td>
</tr>
<tr>
<td>NLDATMMw. Format (p. 250)</td>
<td>Converts a SAS datetime value to the datetime string of the specified locale in the medium representation of the date.</td>
<td></td>
</tr>
<tr>
<td>NLDATMMDw. Format (p. 251)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month and the day of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATMMDLw. Format (p. 252)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the full-length of the month and day of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATMMDMw. Format (p. 253)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using abbreviations.</td>
<td></td>
</tr>
<tr>
<td>NLDATMMDSw. Format (p. 254)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using numbers and delimiters.</td>
<td></td>
</tr>
<tr>
<td>NLDATMMNW. Format (p. 255)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month.</td>
<td></td>
</tr>
<tr>
<td>NLDATMSw. Format (p. 255)</td>
<td>Converts a SAS datetime value to the datetime string of the specified locale in the short representation of the date.</td>
<td></td>
</tr>
<tr>
<td>NLDATMTMW. Format (p. 256)</td>
<td>Converts the time portion of a SAS datetime value to the time-of-day value of the specified locale, and then writes the value as a time of day.</td>
<td></td>
</tr>
<tr>
<td>NLDATMTZW. Format (p. 258)</td>
<td>Converts the time portion of the SAS datetime value to the time of day and time zone of the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NLDATMWW. Format (p. 258)</td>
<td>Converts SAS datetime values to the locale sensitive datetime string as the day of the week and the datetime.</td>
<td></td>
</tr>
<tr>
<td>NLDATMWNW. Format (p. 260)</td>
<td>Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as the day of the week.</td>
<td></td>
</tr>
<tr>
<td>NLDATMWZW. Format (p. 260)</td>
<td>Converts SAS date values of the specified locale to a day-of-week, datetime, and time zone value.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLDATMYMw.</td>
<td>Format (p. 261)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the month and year.</td>
</tr>
<tr>
<td>NLDATMYMLw.</td>
<td>Format (p. 262)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.</td>
</tr>
<tr>
<td>NLDATMYMMw.</td>
<td>Format (p. 263)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.</td>
</tr>
<tr>
<td>NLDATMYMSw.</td>
<td>Format (p. 264)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the month and year with numbers and a delimiter.</td>
</tr>
<tr>
<td>NLDATMYQw.</td>
<td>Format (p. 265)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the quarter and the year.</td>
</tr>
<tr>
<td>NLDATMYQLw.</td>
<td>Format (p. 266)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year’s quarter value (1–4) and the year.</td>
</tr>
<tr>
<td>NLDATMYQMw.</td>
<td>Format (p. 267)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year’s quarter (1–4) and then the year.</td>
</tr>
<tr>
<td>NLDATMYQSw.</td>
<td>Format (p. 268)</td>
<td>Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the quarter (1–4) using numbers and a delimiter.</td>
</tr>
<tr>
<td>NLDATMYRw.</td>
<td>Format (p. 269)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year.</td>
</tr>
<tr>
<td>NLDATMYWw.</td>
<td>Format (p. 270)</td>
<td>Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the week number and the year.</td>
</tr>
<tr>
<td>NLDATMZW.</td>
<td>Format (p. 270)</td>
<td>Converts SAS datetime values to the locale-sensitive datetime string as datetime and time zone.</td>
</tr>
<tr>
<td>NLTIMAPw.</td>
<td>Format (p. 360)</td>
<td>Converts a SAS time value to the time value of a specified locale, and then writes the value as a time value with a.m. or p.m. NLTIMAP also converts SAS date-time values.</td>
</tr>
<tr>
<td>NLTIMEw.</td>
<td>Format (p. 361)</td>
<td>Converts a SAS time value to the time value of the specified locale, and then writes the value as a time value. NLTIME also converts SAS date-time values.</td>
</tr>
<tr>
<td>PDJULGw.</td>
<td>Format (p. 368)</td>
<td>Writes packed Julian date values in the hexadecimal format yyyyddddF for IBM.</td>
</tr>
<tr>
<td>PDJULIw.</td>
<td>Format (p. 369)</td>
<td>Writes packed Julian date values in the hexadecimal format ccyydddF for IBM.</td>
</tr>
<tr>
<td>QTRw.</td>
<td>Format (p. 380)</td>
<td>Writes date values as the quarter of the year.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>QTRRw. Format (p. 381)</td>
<td>Writes date values as the quarter of the year in Roman numerals.</td>
<td></td>
</tr>
<tr>
<td>TIMEw.d Format (p. 402)</td>
<td>Writes time values as hours, minutes, and seconds in the form $hh:mm:ss.ss$.</td>
<td></td>
</tr>
<tr>
<td>TIMEAMPMw.d Format (p. 405)</td>
<td>Writes time and datetime values as hours, minutes, and seconds in the form $hh:mm:ss.ss$ with AM or PM.</td>
<td></td>
</tr>
<tr>
<td>TODw.d Format (p. 406)</td>
<td>Writes SAS time values and the time portion of SAS datetime values in the form $hh:mm:ss.ss$.</td>
<td></td>
</tr>
<tr>
<td>WEEKDATEw. Format (p. 413)</td>
<td>Writes date values as the day of the week and the date in the form day-of-week, month-name dd, yy (or yyyy).</td>
<td></td>
</tr>
<tr>
<td>WEEKDATXw. Format (p. 414)</td>
<td>Writes date values as the day of the week and date in the form day-of-week, dd month-name yy (or yyyy).</td>
<td></td>
</tr>
<tr>
<td>WEEKDAYw. Format (p. 416)</td>
<td>Writes date values as the day of the week.</td>
<td></td>
</tr>
<tr>
<td>WEEKUw. Format (p. 417)</td>
<td>Writes a week number in decimal format by using the U algorithm.</td>
<td></td>
</tr>
<tr>
<td>WEEKVw. Format (p. 419)</td>
<td>Writes a week number in decimal format by using the V algorithm.</td>
<td></td>
</tr>
<tr>
<td>WEEKWw. Format (p. 421)</td>
<td>Writes a week number in decimal format by using the W algorithm.</td>
<td></td>
</tr>
<tr>
<td>WORDDATEw. Format (p. 423)</td>
<td>Writes date values as the name of the month, the day, and the year in the form month-name dd, yyyy.</td>
<td></td>
</tr>
<tr>
<td>WORDDATXw. Format (p. 424)</td>
<td>Writes date values as the day, the name of the month, and the year in the form dd month-name yyyy.</td>
<td></td>
</tr>
<tr>
<td>YEARw. Format (p. 427)</td>
<td>Writes date values as the year.</td>
<td></td>
</tr>
<tr>
<td>YYMMw. Format (p. 429)</td>
<td>Writes date values in the form $&lt;yy&gt;yyMmm$, 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.</td>
<td></td>
</tr>
<tr>
<td>YYMMDDw. Format (p. 430)</td>
<td>Writes date values in the form $yyymmdd$ or $&lt;yy&gt;yy-mm-dd$, where a hyphen is the separator and the year appears as either two or four digits.</td>
<td></td>
</tr>
<tr>
<td>YYMMDDxw. Format (p. 432)</td>
<td>Writes date values in the form $yyymmdd$ or $&lt;yy&gt;yy-mm-dd$, 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.</td>
<td></td>
</tr>
</tbody>
</table>
| YYMMxw. Format (p. 434) | 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 (-),
<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>period (.), slash (/), colon (:), or no separator. The year can be either two or four digits.</td>
</tr>
<tr>
<td>YYMONw. Format (p. 436)</td>
<td></td>
<td>Writes date values in the form yyyy/mm or yyyyymm.</td>
</tr>
<tr>
<td>YYQw. Format (p. 438)</td>
<td></td>
<td>Writes date values in the form &lt;yy&gt;yyQq, where Q is the separator, the year appears as either two or four digits, and q is the quarter of the year.</td>
</tr>
<tr>
<td>YYQcw. Format (p. 439)</td>
<td></td>
<td>Writes date values in the form &lt;yy&gt;yyq or &lt;yy&gt;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.</td>
</tr>
<tr>
<td>YYQRw. Format (p. 441)</td>
<td></td>
<td>Writes date values in the form &lt;yy&gt;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.</td>
</tr>
<tr>
<td>YYQRcw. Format (p. 442)</td>
<td></td>
<td>Writes date values in the form &lt;yy&gt;yyqr or &lt;yy&gt;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.</td>
</tr>
<tr>
<td>YYQZw. Format (p. 444)</td>
<td></td>
<td>Writes SAS date values in the form &lt;yy&gt;&lt;qq&gt;, the year appears as 2 or 4 digits, and qq is the quarter of the year.</td>
</tr>
<tr>
<td>YYWEEKUw. Format (p. 446)</td>
<td></td>
<td>Writes a week number in decimal format by using the U algorithm, excluding day-of-the-week information.</td>
</tr>
<tr>
<td>YYWEEKVw. Format (p. 447)</td>
<td></td>
<td>Writes a week number in decimal format by using the V algorithm, excluding day-of-the-week information.</td>
</tr>
<tr>
<td>YYWEEKWw. Format (p. 449)</td>
<td></td>
<td>Writes a week number in decimal format by using the W algorithm, excluding the day-of-week information.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SN8601Bwd Format (p. 102)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyyymmddThhmmss.</td>
</tr>
<tr>
<td>SN8601BAw.d Format (p. 104)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations yyyyymmdTthhmmss and yyyyymmdTthhmmss.</td>
</tr>
<tr>
<td>SN8601Ew.d Format (p. 106)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyyy-mm-ddTthh:mm:ss.</td>
</tr>
<tr>
<td>SN8601EAw.d Format (p. 107)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddTthh:mm:ss and yyyy-mm-ddTthh:mm:ss.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>$N8601EW:d Format (p. 109)</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>$N8601EXw.d Format (p. 111)</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>$N8601Hw.d Format (p. 113)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms $PnYnMnDnThhMnSn$, dropping omitted components in duration values and using a hyphen (-) for omitted components in datetime values.</td>
</tr>
<tr>
<td>$N8601Xw.d Format (p. 115)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms $PrYnMnDTnHnMnS$ 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.</td>
</tr>
<tr>
<td>B8601DAw. Format (p. 131)</td>
<td></td>
<td>Writes date values by using the ISO 8601 basic notation $yyyyymmd$.</td>
</tr>
<tr>
<td>B8601DNw. Format (p. 132)</td>
<td></td>
<td>Writes dates from datetime values by using the ISO 8601 basic notation $yyyyymmd$.</td>
</tr>
<tr>
<td>B8601DTw.d Format (p. 133)</td>
<td></td>
<td>Writes datetime values by using the ISO 8601 basic notation $yyyyymmdThhmmss&lt;ffffff&gt;$.</td>
</tr>
<tr>
<td>B8601Dxw. Format (p. 135)</td>
<td></td>
<td>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 $yyyyymmdThhmmss+hhmm$.</td>
</tr>
<tr>
<td>B8601DZw. Format (p. 136)</td>
<td></td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone basic notation $yyyyymmdThhmmss+0000$.</td>
</tr>
<tr>
<td>B8601LXw. Format (p. 138)</td>
<td></td>
<td>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 $yyyyymmdThhmmss+_{hhmm}$.</td>
</tr>
<tr>
<td>B8601LZw. Format (p. 139)</td>
<td></td>
<td>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 $hhmmss+_{\pm hhmm}$.</td>
</tr>
<tr>
<td>B8601TMw.d Format (p. 141)</td>
<td></td>
<td>Writes time values by using the ISO 8601 basic notation $hhmmss&lt;fff&gt;$.</td>
</tr>
<tr>
<td>B8601TXw. Format (p. 142)</td>
<td></td>
<td>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 $hhmmss+_{\pm hhmm}$.</td>
</tr>
<tr>
<td>B8601TZw. Format (p. 143)</td>
<td></td>
<td>Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 basic time notation $hhmmss+_{\pm hhmm}$.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>E8601DAw Format (p. 171)</td>
<td>Writes date values by using the ISO 8601 extended notation <code>yyyy-mm-dd</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601DNw Format (p. 172)</td>
<td>Writes dates from SAS datetime values by using the ISO 8601 extended notation <code>yyyy-mm-dd</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601DTw.d Format (p. 173)</td>
<td>Writes datetime values by using the ISO 8601 extended notation <code>yyyy-mm-ddThh:mm:ss.ffffff</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601DXw Format (p. 175)</td>
<td>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 <code>yyyy-mm-ddThh:mm:ss±hh:mm</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601DZw Format (p. 176)</td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) by using the ISO 8601 datetime and time zone extended notation <code>yyyy-mm-ddThh:mm:ss+00:00</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601LXw Format (p. 178)</td>
<td>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 <code>yyyy-mm-ddThh:mm:ss±hh:mm</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601LZw Format (p. 179)</td>
<td>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 <code>hh:mm:ss.&lt;fff&gt;±hh:mm</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601TMw.d Format (p. 181)</td>
<td>Writes time values by using the ISO 8601 extended notation <code>hh:mm:ss.ffffff</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601TXw Format (p. 182)</td>
<td>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 <code>hh:mm:ss±hh:mm</code>.</td>
<td></td>
</tr>
<tr>
<td>E8601TZw.d Format (p. 184)</td>
<td>Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation <code>hh:mm:ss.&lt;fff&gt;±hh:mm</code>.</td>
<td></td>
</tr>
<tr>
<td>Numeric</td>
<td>BESTw Format (p. 126)</td>
<td>SAS chooses the best notation.</td>
</tr>
<tr>
<td></td>
<td>BESTDw.p Format (p. 128)</td>
<td>Prints numeric values, lining up decimal places for values of similar magnitude, and prints integers without decimals.</td>
</tr>
<tr>
<td></td>
<td>BINARYw Format (p. 130)</td>
<td>Converts numeric values to binary representation.</td>
</tr>
<tr>
<td></td>
<td>COMMAw.d Format (p. 145)</td>
<td>Writes numeric values with a comma that separates every three digits and a period that separates the decimal fraction.</td>
</tr>
<tr>
<td></td>
<td>COMMAXw.d Format (p. 147)</td>
<td>Writes numeric values with a period that separates every three digits and a comma that separates the decimal fraction.</td>
</tr>
<tr>
<td></td>
<td>Dw.p Format (p. 148)</td>
<td>Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DOLLARw.d Format (p. 160)</td>
<td>Writes numeric values with a leading dollar sign, a comma that separates every three digits, and a period that separates the decimal fraction.</td>
<td></td>
</tr>
<tr>
<td>DOLLARXw.d Format (p. 161)</td>
<td>Writes numeric values with a leading dollar sign, a period that separates every three digits, and a comma that separates the decimal fraction.</td>
<td></td>
</tr>
<tr>
<td>Ew. Format (p. 170)</td>
<td>Writes numeric values in scientific notation.</td>
<td></td>
</tr>
<tr>
<td>EUROw.d Format (p. 186)</td>
<td>Writes numeric values with a leading euro symbol (E), a comma that separates every three digits, and a period that separates the decimal fraction.</td>
<td></td>
</tr>
<tr>
<td>EUROXw.d Format (p. 189)</td>
<td>Writes numeric values with a leading euro symbol (E), a period that separates every three digits, and a comma that separates the decimal fraction.</td>
<td></td>
</tr>
<tr>
<td>FLOATw.d Format (p. 191)</td>
<td>Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the $d$th power.</td>
<td></td>
</tr>
<tr>
<td>FRACTw. Format (p. 192)</td>
<td>Converts numeric values to fractions.</td>
<td></td>
</tr>
<tr>
<td>HEXw. Format (p. 193)</td>
<td>Converts real binary (floating-point) values to hexadecimal representation.</td>
<td></td>
</tr>
<tr>
<td>IBw.d Format (p. 198)</td>
<td>Writes native integer binary (fixed-point) values, including negative values.</td>
<td></td>
</tr>
<tr>
<td>IBRw.d Format (p. 200)</td>
<td>Writes integer binary (fixed-point) values in Intel and DEC formats.</td>
<td></td>
</tr>
<tr>
<td>IEEEw.d Format (p. 201)</td>
<td>Generates an IEEE floating-point value by multiplying a number by 10 raised to the $d$th power.</td>
<td></td>
</tr>
<tr>
<td>NEGPARENw.d Format (p. 219)</td>
<td>Writes negative numeric values in parentheses.</td>
<td></td>
</tr>
<tr>
<td>NLBESTw. Format (p. 222)</td>
<td>Writes the best numerical notation based on the locale.</td>
<td></td>
</tr>
<tr>
<td>NLMNIAEDw.d Format (p. 271)</td>
<td>Writes the monetary format of the international expression for the United Arab Emirates.</td>
<td></td>
</tr>
<tr>
<td>NLMNIAUDw.d Format (p. 272)</td>
<td>Writes the monetary format of the international expression for Australia.</td>
<td></td>
</tr>
<tr>
<td>NLMNIBGNw.d Format (p. 273)</td>
<td>Writes the monetary format of the international expression for Bulgaria.</td>
<td></td>
</tr>
<tr>
<td>NLMNIBRLw.d Format (p. 274)</td>
<td>Writes the monetary format of the international expression for Brazil.</td>
<td></td>
</tr>
<tr>
<td>NLMNICADw.d Format (p. 275)</td>
<td>Writes the monetary format of the international expression for Canada.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NLMNICHFw.d Format (p. 276)</td>
<td></td>
<td>Writes the monetary format of the international expression for Liechtenstein and Switzerland.</td>
</tr>
<tr>
<td>NLMNICNYw.d Format (p. 277)</td>
<td></td>
<td>Writes the monetary format of the international expression for China.</td>
</tr>
<tr>
<td>NLMNICZKw.d Format (p. 278)</td>
<td></td>
<td>Writes the monetary format of the international expression for the Czech Republic.</td>
</tr>
<tr>
<td>NLMNIDKKw.d Format (p. 279)</td>
<td></td>
<td>Writes the monetary format of the international expression for Denmark, Faroe Island, and Greenland.</td>
</tr>
<tr>
<td>NLMNIEEKw.d Format (p. 280)</td>
<td></td>
<td>Writes the monetary format of the international expression for Estonia.</td>
</tr>
<tr>
<td>NLMNIEGPw.d Format (p. 281)</td>
<td></td>
<td>Writes the monetary format of the international expression for Egypt.</td>
</tr>
<tr>
<td>NLMNIEEURw.d Format (p. 282)</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>NLMNIGBPw.d Format (p. 283)</td>
<td></td>
<td>Writes the monetary format of the international expression for the United Kingdom.</td>
</tr>
<tr>
<td>NLMNIHKDw.d Format (p. 284)</td>
<td></td>
<td>Writes the monetary format of the international expression for Hong Kong.</td>
</tr>
<tr>
<td>NLMNIHRKw.d Format (p. 285)</td>
<td></td>
<td>Writes the monetary format of the international expression for Croatia.</td>
</tr>
<tr>
<td>NLMNIHUFw.d Format (p. 286)</td>
<td></td>
<td>Writes the monetary format of the international expression for Hungary.</td>
</tr>
<tr>
<td>NLMNIIDRw.d Format (p. 287)</td>
<td></td>
<td>Writes the monetary format of the international expression for Indonesia.</td>
</tr>
<tr>
<td>NLMNIILSw.d Format (p. 288)</td>
<td></td>
<td>Writes the monetary format of the international expression for Israel.</td>
</tr>
<tr>
<td>NLMNIINRw.d Format (p. 289)</td>
<td></td>
<td>Writes the monetary format of the international expression for India.</td>
</tr>
<tr>
<td>NLMNIJPYw.d Format (p. 290)</td>
<td></td>
<td>Writes the monetary format of the international expression for Japan.</td>
</tr>
<tr>
<td>NLMNIKR Rw.d Format (p. 291)</td>
<td></td>
<td>Writes the monetary format of the international expression for South Korea.</td>
</tr>
<tr>
<td>NLMNILTLw.d Format (p. 292)</td>
<td></td>
<td>Writes the monetary format of the international expression for Lithuania.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NLMNIVLw.d Format (p. 293)</td>
<td>Writes the monetary format of the international expression for Latvia.</td>
<td></td>
</tr>
<tr>
<td>NLMNIMOPw.d Format (p. 294)</td>
<td>Writes the monetary format of the international expression for Macau.</td>
<td></td>
</tr>
<tr>
<td>NLMNIMNXw.d Format (p. 295)</td>
<td>Writes the monetary format of the international expression for Mexico.</td>
<td></td>
</tr>
<tr>
<td>NLMNIMYRw.d Format (p. 296)</td>
<td>Writes the monetary format of the international expression for Malaysia.</td>
<td></td>
</tr>
<tr>
<td>NLMNINOKw.d Format (p. 297)</td>
<td>Writes the monetary format of the international expression for Norway.</td>
<td></td>
</tr>
<tr>
<td>NLMNINZDw.d Format (p. 298)</td>
<td>Writes the monetary format of the international expression for New Zealand.</td>
<td></td>
</tr>
<tr>
<td>NLMNIPLNw.d Format (p. 299)</td>
<td>Writes the monetary format of the international expression for Poland.</td>
<td></td>
</tr>
<tr>
<td>NLMNIRUBw.d Format (p. 300)</td>
<td>Writes the monetary format of the international expression for Russia.</td>
<td></td>
</tr>
<tr>
<td>NLMNISEKW.d Format (p. 301)</td>
<td>Writes the monetary format of the international expression for Sweden.</td>
<td></td>
</tr>
<tr>
<td>NLMNISGDDw.d Format (p. 302)</td>
<td>Writes the monetary format of the international expression for Singapore.</td>
<td></td>
</tr>
<tr>
<td>NLMNITHBw.d Format (p. 303)</td>
<td>Writes the monetary format of the international expression for Thailand.</td>
<td></td>
</tr>
<tr>
<td>NLMNITRYw.d Format (p. 304)</td>
<td>Writes the monetary format of the international expression for Turkey.</td>
<td></td>
</tr>
<tr>
<td>NLMNITWDw.d Format (p. 305)</td>
<td>Writes the monetary format of the international expression for Taiwan.</td>
<td></td>
</tr>
<tr>
<td>NLMNIUSDw.d Format (p. 306)</td>
<td>Writes the monetary format of the international expression for Puerto Rico and the United States.</td>
<td></td>
</tr>
<tr>
<td>NLMNIZARw.d Format (p. 307)</td>
<td>Writes the monetary format of the international expression for South Africa.</td>
<td></td>
</tr>
<tr>
<td>NLMNLAEDx.d Format (p. 308)</td>
<td>Writes the monetary format of the local expression for the United Arab Emirates.</td>
<td></td>
</tr>
<tr>
<td>NLMNLAUDw.d Format (p. 309)</td>
<td>Writes the monetary format of the local expression for Australia.</td>
<td></td>
</tr>
<tr>
<td>NLMNLBGNw.d Format (p. 310)</td>
<td>Writes the monetary format of the local expression for Bulgaria.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>NLMNLBL</td>
<td>$w.d$ Format (p. 311)</td>
<td>Writes the monetary format of the local expression for Brazil.</td>
</tr>
<tr>
<td>NLMNLCA</td>
<td>$d$ Format (p. 312)</td>
<td>Writes the monetary format of the local expression for Canada.</td>
</tr>
<tr>
<td>NLMNLCHF</td>
<td>$w.d$ Format (p. 313)</td>
<td>Writes the monetary format of the local expression for Liechtenstein and Switzerland.</td>
</tr>
<tr>
<td>NLMNLCNY</td>
<td>$w.d$ Format (p. 314)</td>
<td>Writes the monetary format of the local expression for China.</td>
</tr>
<tr>
<td>NLMNLZK</td>
<td>$w.d$ Format (p. 315)</td>
<td>Writes the monetary format of the local expression for the Czech Republic.</td>
</tr>
<tr>
<td>NLMNLDKK</td>
<td>$w.d$ Format (p. 316)</td>
<td>Writes the monetary format of the local expression for Denmark, Faroe Island, and Greenland.</td>
</tr>
<tr>
<td>NLMNLEEK</td>
<td>$w.d$ Format (p. 317)</td>
<td>Writes the monetary format of the local expression for Estonia.</td>
</tr>
<tr>
<td>NLMNLEG</td>
<td>$w.d$ Format (p. 318)</td>
<td>Writes the monetary format of the local expression for Egypt.</td>
</tr>
<tr>
<td>NLMNLEUR</td>
<td>$w.d$ Format (p. 319)</td>
<td>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.</td>
</tr>
<tr>
<td>NLMNGBP</td>
<td>$w.d$ Format (p. 320)</td>
<td>Writes the monetary format of the local expression for the United Kingdom.</td>
</tr>
<tr>
<td>NLMNLHKD</td>
<td>$w.d$ Format (p. 321)</td>
<td>Writes the monetary format of the local expression for Hong Kong.</td>
</tr>
<tr>
<td>NLMNLHKR</td>
<td>$w.d$ Format (p. 322)</td>
<td>Writes the monetary format of the local expression for Croatia.</td>
</tr>
<tr>
<td>NLMNLU</td>
<td>$w.d$ Format (p. 323)</td>
<td>Writes the monetary format of the local expression for Hungary.</td>
</tr>
<tr>
<td>NLMNLIDR</td>
<td>$w.d$ Format (p. 324)</td>
<td>Writes the monetary format of the local expression for Indonesia.</td>
</tr>
<tr>
<td>NLMNLS</td>
<td>$w.d$ Format (p. 325)</td>
<td>Writes the monetary format of the local expression for Israel.</td>
</tr>
<tr>
<td>NLMNLINR</td>
<td>$w.d$ Format (p. 326)</td>
<td>Writes the monetary format of the local expression for India.</td>
</tr>
<tr>
<td>NLMNJPY</td>
<td>$w.d$ Format (p. 327)</td>
<td>Writes the monetary format of the international expression for Japan.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLMNLKRWw.d</td>
<td>Format (p. 328)</td>
<td>Writes the monetary format of the local expression for South Korea.</td>
</tr>
<tr>
<td>NLMNLLTLw.d</td>
<td>Format (p. 329)</td>
<td>Writes the monetary format of the local expression for Lithuania.</td>
</tr>
<tr>
<td>NLMNLLVLw.d</td>
<td>Format (p. 330)</td>
<td>Writes the monetary format of the local expression for Latvia.</td>
</tr>
<tr>
<td>NLMNLMOPw.d</td>
<td>Format (p. 331)</td>
<td>Writes the monetary format of the local expression for Macau.</td>
</tr>
<tr>
<td>NLMNLMXNw.d</td>
<td>Format (p. 332)</td>
<td>Writes the monetary format of the local expression for Mexico.</td>
</tr>
<tr>
<td>NLMNLMYRw.d</td>
<td>Format (p. 333)</td>
<td>Writes the monetary format of the local expression for Malaysia.</td>
</tr>
<tr>
<td>NLMNLOKw.d</td>
<td>Format (p. 334)</td>
<td>Writes the monetary format of the local expression for Norway.</td>
</tr>
<tr>
<td>NLMNLNZDw.d</td>
<td>Format (p. 335)</td>
<td>Writes the monetary format of the local expression for New Zealand.</td>
</tr>
<tr>
<td>NLMNPLNw.d</td>
<td>Format (p. 336)</td>
<td>Writes the monetary format of the local expression for Poland.</td>
</tr>
<tr>
<td>NLMNLRUBw.d</td>
<td>Format (p. 337)</td>
<td>Writes the monetary format of the local expression for Russia.</td>
</tr>
<tr>
<td>NLMNLEKw.d</td>
<td>Format (p. 338)</td>
<td>Writes the monetary format of the local expression for Sweden.</td>
</tr>
<tr>
<td>NLMNLUSDw.d</td>
<td>Format (p. 339)</td>
<td>Writes the monetary format of the local expression for Singapore.</td>
</tr>
<tr>
<td>NLMNLTHBw.d</td>
<td>Format (p. 340)</td>
<td>Writes the monetary format of the local expression for Thailand.</td>
</tr>
<tr>
<td>NLMNLTRYw.d</td>
<td>Format (p. 341)</td>
<td>Writes the monetary format of the local expression for Turkey.</td>
</tr>
<tr>
<td>NLMNLUSDw.d</td>
<td>Format (p. 342)</td>
<td>Writes the monetary format of the local expression for Taiwan.</td>
</tr>
<tr>
<td>NLMNLTHBw.d</td>
<td>Format (p. 343)</td>
<td>Writes the monetary format of the local expression for Puerto Rico and the United States.</td>
</tr>
<tr>
<td>NLMNLZARw.d</td>
<td>Format (p. 344)</td>
<td>Writes the monetary format of the local expression for South Africa.</td>
</tr>
<tr>
<td>NLMNYw.d</td>
<td>Format (p. 345)</td>
<td>Writes the monetary format of the local expression in the specified locale using local currency.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NLMNYIw.d Format (p. 346)</td>
<td>Writes the monetary format of the international expression in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NLMUw.d Format (p. 347)</td>
<td>Writes the numeric format of the local expression in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NNUMIw.d Format (p. 349)</td>
<td>Writes the numeric format of the international expression in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NLPCTw.d Format (p. 350)</td>
<td>Writes percentage data of the local expression in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NLPCTIw.d Format (p. 352)</td>
<td>Writes percentage data of the international expression in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NLPCTNW.d Format (p. 353)</td>
<td>Produces percentages, using a minus sign for negative values.</td>
<td></td>
</tr>
<tr>
<td>NLPCTPW.d Format (p. 354)</td>
<td>Writes locale-specific numeric values as percentages.</td>
<td></td>
</tr>
<tr>
<td>NLPVALEW.d Format (p. 355)</td>
<td>Writes p-values of the local expression in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NLSTRMNONw.d Format (p. 356)</td>
<td>Writes the month name in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NLSTROQTRW.d Format (p. 357)</td>
<td>Writes a numeric value as the quarter-of-the-year in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NLSTRWKW.d Format (p. 359)</td>
<td>Writes a numeric value as the day-of-the-week in the specified locale.</td>
<td></td>
</tr>
<tr>
<td>NUMXW.d Format (p. 362)</td>
<td>Writes numeric values with a comma in place of the decimal point.</td>
<td></td>
</tr>
<tr>
<td>OCTALW. Format (p. 364)</td>
<td>Converts numeric values to octal representation.</td>
<td></td>
</tr>
<tr>
<td>OODSRW.d Format (p. 364)</td>
<td>Writes odds ratios.</td>
<td></td>
</tr>
<tr>
<td>PDW.d Format (p. 366)</td>
<td>Writes data in packed decimal format.</td>
<td></td>
</tr>
<tr>
<td>PERCENTW.d Format (p. 371)</td>
<td>Writes numeric values as percentages.</td>
<td></td>
</tr>
<tr>
<td>PERCENTNW.d Format (p. 372)</td>
<td>Produces percentages, using a minus sign for negative values.</td>
<td></td>
</tr>
<tr>
<td>PIBW.d Format (p. 374)</td>
<td>Writes positive integer binary (fixed-point) values.</td>
<td></td>
</tr>
<tr>
<td>PIBRW.d Format (p. 376)</td>
<td>Writes positive integer binary (fixed-point) values in Intel and DEC formats.</td>
<td></td>
</tr>
<tr>
<td>PKW.d Format (p. 377)</td>
<td>Writes data in unsigned packed decimal format.</td>
<td></td>
</tr>
<tr>
<td>PVALUEW.d Format (p. 378)</td>
<td>Writes p-values.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>RBw.d Format (p. 382)</td>
<td>Writes real binary data (floating-point) in real binary format.</td>
<td></td>
</tr>
<tr>
<td>ROMANw. Format (p. 383)</td>
<td>Writes numeric values as Roman numerals.</td>
<td></td>
</tr>
<tr>
<td>S370FFw.d Format (p. 384)</td>
<td>Writes native standard numeric data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FIBw.d Format (p. 385)</td>
<td>Writes integer binary (fixed-point) values, including negative values, in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FIBUw.d Format (p. 387)</td>
<td>Writes unsigned integer binary (fixed-point) values in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPDw.d Format (p. 389)</td>
<td>Writes packed decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPDUw.d Format (p. 390)</td>
<td>Writes unsigned packed decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPIBw.d Format (p. 391)</td>
<td>Writes positive integer binary (fixed-point) values in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FRBw.d Format (p. 393)</td>
<td>Writes real binary (floating-point) data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDw.d Format (p. 394)</td>
<td>Writes zoned decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDLw.d Format (p. 396)</td>
<td>Writes zoned decimal leading-sign data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDSw.d Format (p. 397)</td>
<td>Writes zoned decimal separate leading-sign data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDTw.d Format (p. 399)</td>
<td>Writes zoned decimal separate trailing-sign data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDUw.d Format (p. 400)</td>
<td>Writes unsigned zoned decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>SSNw. Format (p. 401)</td>
<td>Writes Social Security numbers.</td>
<td></td>
</tr>
<tr>
<td>VAXRBw.d Format (p. 409)</td>
<td>Writes real binary (floating-point) data in VMS format.</td>
<td></td>
</tr>
<tr>
<td>VMSZNw.d Format (p. 410)</td>
<td>Generates VMS and MicroFocus COBOL zoned numeric data.</td>
<td></td>
</tr>
<tr>
<td>w.d Format (p. 411)</td>
<td>Writes standard numeric data one digit per byte.</td>
<td></td>
</tr>
<tr>
<td>WORDFw. Format (p. 425)</td>
<td>Writes numeric values as words with fractions that are shown numerically.</td>
<td></td>
</tr>
<tr>
<td>WORDSw. Format (p. 426)</td>
<td>Writes numeric values as words.</td>
<td></td>
</tr>
<tr>
<td>YYMONw. Format (p. 436)</td>
<td>Writes date values in the form yyymm or yyyyymm.</td>
<td></td>
</tr>
</tbody>
</table>
| YYQRw. Format (p. 442) | 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.

\[
Zw.d \text{ Format (p. 450)}
\]

Writes standard numeric data with leading 0s.

\[
ZDw.d \text{ Format (p. 451)}
\]

Writes numeric data in zoned decimal format.

### 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

\[
$ASCIIw.
\]

### 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, $ASCIIw$. converts EBCDIC character data to ASCIIw.
- On all other systems, $ASCIIw$. behaves like the $CHARw$. format.

### Example

```plaintext
data one;
   input x $;
   datalines;
   abc
   ABC
```
SAS writes the following results to the log:

```plaintext
abc
ABC
()*
```

### $\text{BASE64Xw}$ Format

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

**Categories:** CAS  
Character

**Alignment:** Left

### Syntax

$\text{BASE64Xw}$

### Syntax Description

\(w\)

specifies the width of the output field.

You can use the following formula to determine the width:

\[
\text{format-width} = \frac{\text{variable-length} + 2}{3} \times 4
\]

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 \((48+2)/3 \times 4=64\).

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;

RkNBMDFBNzk5M0JDICAgICAgICAgICAgICAgICAgICAgTXlQYXNzd29yZCAgICAgICAgICAgICAgICAgICAg

d3d3Lm15ZG9tYWluLmNvbS9teWhpZGRlblVSTCAg

See Also

- The LIBNAME statement option “XMLDOUBLE=DISPLAY | INTERNAL” in SAS XMLV2 and XML LIBNAME Engines: User’s Guide

Informats:

- “$BASE64Xw. Informat” on page 489

$BINARYw. Format

Converts character data to binary representation.

**Categories:** CAS

Character

**Alignment:** Left

Syntax

$BINARYw:
**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 $\text{BINARY}w$. format converts character values to binary representation. The $\text{BINARY}w$. format converts numeric values to binary representation.

**Example: Converting Two Characters to Binary Representation**

```sas
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.

---

**$\text{CHAR}w$. Format**

Writes standard character data.

**Categories:**

- CAS
- Character

**Alignment:**

Left

**Syntax**

$\text{CHAR}w$.

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

<table>
<thead>
<tr>
<th>Language</th>
<th>Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>$CHAR8.</td>
</tr>
<tr>
<td>C</td>
<td>char [8]</td>
</tr>
<tr>
<td>COBOL</td>
<td>PIC x(8)</td>
</tr>
</tbody>
</table>

Example:

```sas
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**

$CSTRw.
**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**

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x $cstr. y;</td>
<td>XYZ*</td>
<td>XYZ</td>
</tr>
</tbody>
</table>

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

---

**$EBCDIC\_w$. 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
Details

If EBCDIC is the native format, no conversion occurs.

On ASCII systems, the $EBCDICw. 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 $EBCDICw. format is Open ed-1047. If the locale is de_DE (German_Germany), the default encoding that is used by the $EBCDICw. 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= System Option” in SAS National Language Support (NLS): Reference Guide.

Comparisons

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

Examples

Example 1

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

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Locale Value Is en_US</th>
<th>Locale Value Is fr_FR</th>
<th>Locale Value Is de_DE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABC</td>
<td>C1C2C3</td>
<td>C1C2C3</td>
<td>C1C2C3</td>
</tr>
<tr>
<td>{ä}</td>
<td>C043D0</td>
<td>514354</td>
<td>43C0DC</td>
</tr>
<tr>
<td>[@]</td>
<td>AD7CBD</td>
<td>9044B5</td>
<td>63B5FC</td>
</tr>
</tbody>
</table>
Example 2
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. Format: Windows” in SAS Companion for Windows

Syntax
$HEXw.

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, $HEXw. pads it with blanks.

Details
The $HEXw. format converts each character to two hexadecimal characters. Each blank counts as one character, including trailing blanks.

Comparisons
The HEXw. format converts real binary numbers to their hexadecimal equivalent.

Example
data one;
  x='AB';
  put x $hex4.;
run;
### $\text{MSGCASE}w$. 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

`$\text{MSGCASE}w$.`

#### Syntax Description

`\text{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

```sas
data one;
  x='sas';
  put x $msgcase.;
run;
```

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>sas</td>
<td>SAS</td>
</tr>
</tbody>
</table>
See Also

System Options:

• “MSGCASE System Option: UNIX” in SAS Companion for UNIX Environments
• “MSGCASE System Option: Windows” in SAS Companion for Windows
• “MSGCASE System Option: z/OS” in SAS Companion for z/OS

$\text{N8601Bw}.d$ Format

Writes ISO 8601 duration, datetime, and interval forms by using the basic notations $PnYnMnDTnHnMnS$ and $yyyymmddThhmmss$.

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

$\text{SN8601Bw}.d$

Syntax Description

\(w\)

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1–200</td>
</tr>
</tbody>
</table>

\(d\)

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0–6</td>
</tr>
</tbody>
</table>

Details

The $\text{SN8601B}$ format writes ISO 8601 duration, datetime, and interval values as character data for the following basic notations:

• $PnYnMnDTnHnMnS$
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.

```plaintext
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mynew $n8601b.;</td>
<td>20180608094105F01</td>
<td>20180608T09410596</td>
</tr>
</tbody>
</table>

**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.

```plaintext
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dur $n8601b.;</td>
<td>0001214013432FFC</td>
<td>P1Y2M14DT1H34M32S</td>
</tr>
</tbody>
</table>
See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

$N8601BAw.d Format

Writes ISO 8601 duration, datetime, and interval forms by using the basic notations $PyyyyymmddThhmmss$ and $yyyyymmddThhmmss$.

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 5.5.4.2, alternative format

Syntax

$N8601BAw.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:

- $PyyyyymmddThhmmss$
- $yyyyymmddThhmmss$
- $PyyyyymmddThhmmss/yyyyymmddThhmmss$
- $yyyyymmddThhmmss/PyyyyymmddThhmmss$
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mynew $n8601ba.;</td>
<td>2018608094105F01</td>
<td>20180608T09410596</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dur $n8601ba.;</td>
<td>0001214013432FFC</td>
<td>P0001214013432F0C</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40
$N8601Ew.d$ Format

Writes ISO 8601 duration, datetime, and interval forms by using the extended notations $PnYnMnDTnHnMnS$ and $yyyy-mm-ddThh:mm:ss$.

**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**

$N8601Ew.d$

**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:

- $PnYnMnDTnHnMnS$
- $yyyy-mm-ddThh:mm:ss$
- $PnYnMnDTnHnMnS/yyyy-mm-ddThh:mm:ss$
- $yyyy-mm-ddThh:mm:ssT/PnYnMnDTnHnMnS$

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.

```sas
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mynew $n8601e.;</td>
<td>2018608094105F01</td>
<td>2018-06-08T09:41:05.96</td>
</tr>
</tbody>
</table>

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.

```sas
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);   /* 2 */
   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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dur $n8601e.;</td>
<td>0001214013432FFC</td>
<td>P1Y2M14DT1H34M32S</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

$N8601EAw.d Format

Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh: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
$N8601EAw.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:

- $Pyyyy-mm-ddThh:mm:ss$
- $yyyy-mm-ddThh:mm:ss$
- $Pyyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss$
- $yyyy-mm-ddThh:mm:ss/Pyyyy-mm-ddThh:mm:ss$

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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mynew $n8601ea.;</td>
<td>20180608094105F01</td>
<td>2018-06-08T09:41:05.96</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dur $n8601ea.;</td>
<td>P0001-02-14T01:34:32</td>
<td>P1Y2M14DT1H34M32S</td>
</tr>
</tbody>
</table>

**See Also**

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40
Syntax

$N8601EHw.d

Syntax Description

$N8601EHw

$N8601EHw 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:

- `Pyyyy-mm-ddThh:mm:ss`
- `yyyy-mm-ddThh:mm:ss`
- `Pyyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss`
- `yyyy-mm-ddThh:mm:ss/Pyyyy-mm-ddThh:mm:ss`
- `yyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh: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.

```plaintext
data one;
lengh 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.
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 (-).

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dur $n8601eh.;</td>
<td>0001214013432FFC</td>
<td>P0001---14T01:34:32</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

$N8601EXw.d Format

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.

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

$N8601EXw.d

Syntax Description

w
  specifies the width of the output field.
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:

• $Pyyyy-mm-ddThh:mm:ss$
• $yyyy-mm-ddThh:mm:ss$
• $Pyyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss$
• $yyyy-mm-ddThh:mm:ss/Pyyyy-mm-ddThh:mm:ss$
• $yyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss$

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.

```plaintext
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mynew $n8601ex.;</td>
<td>20180608094105F01</td>
<td>2018-06-08T09:41:05.9</td>
</tr>
</tbody>
</table>
**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.

```plaintext
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`.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dur $n8601ex.;</td>
<td>0001214013432FFC</td>
<td>P0001-02-14T01:34:3</td>
</tr>
</tbody>
</table>

**See Also**
“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

**$N8601Hw.d Format**

Writes ISO 8601 duration, datetime, and interval forms \(P_nY_nM_nDT_nH_nM_nS\) and \(yyyy-mm-ddThh:mm:ss\), 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**

\( $N8601Hw.d \)

**Syntax Description**

- `w` specifies the width of the output field.

  Default 50
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 PnYnMnDTnHnMnS form and using a hyphen (-) to represent omitted components in the datetime form:

- PnYnMnDTnHnMnS
- yyyy-mm-ddThh:mm:ss
- PnYnMnDTnHnMnS/yyyy-mm-ddThh:mm:ss
- yyyy-mm-ddThh:mm:ss/T/PnYnMnDTnHnMnS
- yyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh: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.

```plaintext
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mynew $n8601e.;</td>
<td>2018608094105F01</td>
<td>2018-06-08T09:41:05.96</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dur $n8601h.;</td>
<td>0001214013432FFC</td>
<td>P1Y2M14DT1H34M32S</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

$N8601Xw.d Format

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.

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

$N8601Xw.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.
Details

The SN8601X format writes ISO 8601 durations, intervals, and datetimes in the following forms, omitting components in the \( PnYnMnDTnHnMnS \) form and using an \( x \) to represent omitted components in the datetime form:

- \( PnYnMnDTnHnMnS \)
- \( yyyy-mm-ddThh:mm:ss \)
- \( PnYnMnDTnHnMnS/yyyy-mm-ddThh:mm:ss \)
- \( yyyy-mm-ddThh:mm:ss/TpYnMnDTnHnMnS \)
- \( yyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss \)

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.

```plaintext
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.;
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 \( x \).

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dt $n8601x.;</td>
<td>20186FF101520001</td>
<td>2018-06-xxT10:15:20</td>
</tr>
<tr>
<td>put dt $n8601x.;</td>
<td>2018F05101520001</td>
<td>2018-xx-05T10:15:20</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

$OCTALw. Format

Converts character data to octal representation.
$OCTAL_w$. Format

Syntax Description

\textbf{$w$}

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 \texttt{$OCTAL_w$.} format converts character values to the octal representation of their character codes. The \texttt{OCTAL} format converts numeric values to octal representation.

Example

The following example shows ASCII output when you use the \texttt{$OCTAL_w$.} format.

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

SAS writes the following results to the log.

```
14116216400040
162151343145040
14214156153040
```

$QUOTE_w$. Format

Writes data values that are enclosed in double quotation marks.

**Categories:**
CAS
Character
**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\"B\"".

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;

<table>
<thead>
<tr>
<th>Value of original</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>&quot;SAS&quot;</td>
</tr>
<tr>
<td>SAS's</td>
<td>&quot;SAS's&quot;</td>
</tr>
<tr>
<td>'ad&quot;verb&quot;&quot;</td>
<td>&quot;'ad&quot;''verb&quot;&quot;</td>
</tr>
<tr>
<td>'ad''verb'</td>
<td>&quot;'ad''verb'&quot;</td>
</tr>
<tr>
<td>&quot;ad&quot;''verb&quot;</td>
<td>&quot;'&quot;&quot;ad&quot;''verb&quot;&quot;</td>
</tr>
<tr>
<td>deoxyribonucleotide</td>
<td><em>deoxyribonucleotide</em> *</td>
</tr>
</tbody>
</table>

* 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

$REVERJw.$

Syntax Description

$w$

specifies the width of the output field.

Default 1, if $w$ is not specified

Range 1–32767

Comparisons

The $REVERJw.$ format is similar to the $REVERSw.$ format except that $REVERSw.$ left-aligns the result by trimming all leading blanks.

Example

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x $reverj7.;</td>
<td>ABCD###</td>
<td>DCBA</td>
</tr>
<tr>
<td>put y $reverj7.;</td>
<td>###ABCD</td>
<td>DCBA</td>
</tr>
</tbody>
</table>

* The character # represents a blank space.

$REVERSw.$ Format

Writes character data in reverse order and left-aligns

Categories: CAS Character

Alignment: Left

Syntax

$REVERSw.$
Syntax Description

\( w \)

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>1 if ( w ) is not specified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1–32767</td>
</tr>
</tbody>
</table>

Comparisons

The \$REVESw. format is similar to the \$REVERSJw. format except that \$REVESJw. does not left-align the result.

Example

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

<table>
<thead>
<tr>
<th>Statement</th>
<th>Name*</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x $revers7.;</td>
<td>ABCD###</td>
<td>DCBA</td>
</tr>
<tr>
<td>put y $revers7.;</td>
<td>###ABCD</td>
<td>DCBA</td>
</tr>
</tbody>
</table>

* The character # represents a blank space.

$UPCASEw. Format

Converts character data to uppercase.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Character</td>
</tr>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
</tbody>
</table>

Syntax

\$UPCASEw.

Syntax Description

\( w \)

specifies the width of the output field.
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;
```

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>coxe-ryan</td>
<td>COXE-RYAN</td>
</tr>
</tbody>
</table>

$UUIDw. Format

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

Categories: CAS
Character

Alignment: Left

Syntax
```
$UUIDw.
```

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;

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1548611fb8cb6a47a721a6fd284e74f2</td>
<td>1548611f-b8cb-6a47-a721-a6fd284e74f2</td>
</tr>
</tbody>
</table>

$VARYINGw. Format

Writes character data of varying length.

Valid in:  in DATA step
Category:  Character
Alignment:  Left
Restriction:  This format is not supported in a DATA step that runs in CAS.

Syntax

$VARYINGw. 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 $VARYINGw. 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 $V A R Y I N G w$, when the length of a character value differs from record to record. After writing a data value with $V A R Y I N G w$, the pointer's position is the first column after the value.

Examples

**Example 1: Obtaining a Variable Length Directly**

```sas
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:

```sas
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.

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York 8</td>
<td>New York</td>
</tr>
<tr>
<td>Toronto 7</td>
<td>Toronto</td>
</tr>
<tr>
<td>Buenos Aires 12</td>
<td>Buenos Aires</td>
</tr>
</tbody>
</table>
Value of name *  Result

Tokyo 5  Tokyo

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

**Example 2: Obtaining a Variable Length Indirectly**

Use the LENGTH function to determine the length of a variable. The data values and the results follow the explanation of these SAS statements:

```sas
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.

<table>
<thead>
<tr>
<th>Value of varlen*</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York</td>
<td>New York</td>
</tr>
<tr>
<td>Toronto</td>
<td>Toronto</td>
</tr>
<tr>
<td>Buenos Aires</td>
<td>Buenos Aires</td>
</tr>
<tr>
<td>Tokyo</td>
<td>Tokyo</td>
</tr>
</tbody>
</table>

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

**$w. Format**

Writes standard character data.

- **Category:** Character
- **Alignment:** Left
- **Alias:** $Fw.

**Syntax**

```
$w.
```

**Syntax Description**

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

  - **Default**
    - 1, if the length of the variable is undefined. Otherwise, the default is the length of the variable.
Comparisons

The $w$. format is not identical to the $\text{CHAR}w$. format. The $w$. format trims trailing blanks. The $\text{CHAR}w$. format does not trim trailing blanks.

The $w$. format and the $\text{CHAR}w$. 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 $\text{char}5. ;
  datalines;
  Cary
  Tokyo
  ;
run;
data two;
  set one;
  put @name $5. ;
  put name $ 10-15;
run;

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>#Cary</td>
<td>Cary</td>
</tr>
<tr>
<td>Tokyo</td>
<td>Tokyo</td>
</tr>
</tbody>
</table>

* 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= System Option” in SAS System Options: Reference.
Syntax

\texttt{BESTw.}

\textbf{Syntax Description}

\texttt{w}

specifies the width of the output field.

\begin{tabular}{|l|l|}
\hline
\textbf{Default} & 12 \\
\hline
\textbf{Range} & 1–32 \\
\hline
\textbf{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. \\
\hline
\end{tabular}

\textbf{Details}

When a format is not specified for writing a numeric value, SAS uses the \texttt{BESTw.} format as the default format. The \texttt{BESTw.} 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 \texttt{BESTw.} format is affected by the setting of the \texttt{DECIMALCONV} option. For more information, see “\texttt{DECIMALCONV=} System Option” in \textit{SAS System Options: Reference}. These rules are generally applicable regardless of the option setting.

\textbf{Comparisons}

- The \texttt{BESTw.} 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 \texttt{Dw.p} format writes numbers with the desired precision and more alignment than the \texttt{BESTw.} format.
- The \texttt{BESTDw.p} format is a combination of the \texttt{BESTw.} format and the \texttt{Dw.p} format in that it formats all numeric data, and it does a better job of aligning decimals than the \texttt{BESTw.} format.
• The \( w.d \) format aligns decimal points, if possible, but does not necessarily show the same precision for all numbers.

**Example**

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

The following statements produce these results:

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=1257000;</td>
<td>1.26E6</td>
</tr>
<tr>
<td>put x best6.;</td>
<td>1.26E6</td>
</tr>
<tr>
<td>put x best3.;</td>
<td>1E6</td>
</tr>
<tr>
<td>x=1257000;</td>
<td>1E6</td>
</tr>
</tbody>
</table>

**See Also**

Formats:

• “BESTD\(w.p\) Format” on page 128

**BESTD\(w.p\) 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= System Option” in SAS System Options: Reference.

**Syntax**

```
BESTD\(w.p\)
```

**Syntax Description**

\( w\)

specifies the width of the output field.
**BESTDw.p Format**

**Default**

12

**Range**

1–32

$p$

specifies the precision. This argument is optional.

<table>
<thead>
<tr>
<th>Default</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0 to $w$–1</td>
</tr>
<tr>
<td>Requirement</td>
<td>$p$ must be less than $w$.</td>
</tr>
<tr>
<td>Tip</td>
<td>If $p$ is omitted or is specified as 0, then $p$ is set to 3.</td>
</tr>
</tbody>
</table>

**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 Dw.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 Dw.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**

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

data _null_;  
  set one;
  put x bestd9.;
run;
```
### Data Line

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>12345</td>
</tr>
<tr>
<td>123.45</td>
<td>123.4500</td>
</tr>
<tr>
<td>1.2345</td>
<td>1.2345</td>
</tr>
<tr>
<td>.12345</td>
<td>0.12345</td>
</tr>
</tbody>
</table>

### See Also

**Formats:**
- “BESTw. Format” on page 126
- “Dw.p Format” on page 148

### 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

```
BINARYw.
```

#### Syntax Description

- `w` specifies the width of the output field.

  - **Default:** 8
  - **Range:** 1–64

#### Comparisons

BINARYw. converts numeric values to binary representation. The $BINARYw. format converts character values to binary representation.

#### Example

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

```
datalines;
123.45
123
-123
;
run;

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.45</td>
<td>01111011</td>
</tr>
<tr>
<td>123</td>
<td>01111011</td>
</tr>
<tr>
<td>-123</td>
<td>10000101</td>
</tr>
</tbody>
</table>

**B8601DAw. Format**

Writes date values by using the ISO 8601 basic notation `yyyyymmdd`.

- **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 5.2.1.1, complete representation

**Syntax**

```
B8601DAw.
```

**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:

yyy
  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;

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>21319</td>
<td>20180515</td>
</tr>
</tbody>
</table>

See Also
“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations”
on page 40

B8601DNw. Format

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

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 5.2.1.1, complete representation

Syntax

B8601DNw.

Syntax Description

w
  specifies the width of the output field.

Default 10
Details
The B8601DN format writes the date from a datetime value by using the ISO 8601 basic
date notation yyyyymmdd:

*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

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>21319</td>
<td>20180515</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations”
on page 40

B8601DTw.d Format

Writes datetime values by using the ISO 8601 basic notation yyyyymmddThhmmss<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= System Option” in SAS System Options:
Reference.

**Supports:** ISO 8601 Element 5.4.1, complete representation

**Syntax**

B8601DTw.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 $\text{yyyy}mmddTHhmmss<ffffff>$:

\- \text{yyyy} is a four-digit year.
\- \text{mm} is a two-digit month (zero padded) between 01 and 12.
\- \text{dd} is a two-digit day of the month (zero padded) between 01 and 31.
\- \text{hh} is a two-digit hour (zero padded) between 00 and 23.
\- \text{mm} is a two-digit minute (zero padded) between 00 and 59.
\- \text{ss} is a two-digit second (zero padded) between 00 and 59.
\- \text{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;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1842015142</td>
<td>20180515T145222</td>
</tr>
</tbody>
</table>
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 `yyyyymmddThhmmss +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 Element 5.4.1, complete representation

### Syntax

`B8601DXw.`

### 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:

- `yyyyymmddThhmmss+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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data <em>null</em>; t='01Feb2013T12:34:56'dt ; put t b8601dx.; run;</td>
<td>20130201T073456-0500</td>
</tr>
<tr>
<td>options timezone='America/Adak'; data <em>null</em>; t='01Feb2013T12:34:56'dt ; put t b8601dx.; run;</td>
<td>20130201T023456-1000</td>
</tr>
</tbody>
</table>

**See Also**

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

**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 yyyymmddTthhmmss+0000.

**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 Element 5.4.1, complete representation

**Syntax**

B8601DZw.
**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:

- **yyyymmdTthmmss+0000**
  - *Note*: Use this form when \( w \) is large enough to support this time zone notation.

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

- **yyy**
  - 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. Format” on page 139.

**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**

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

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

**See Also**

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

---

### 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 `yyyyymmddThhmmss+|--hhmm`.

- **Categories:** Date and Time
- **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

```
B8601LXw
```

#### 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:

- `yyyyymmddThhmmss+|--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.
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:

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

<table>
<thead>
<tr>
<th>Value of blx</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1675341296</td>
<td>20130201T123456-0500</td>
</tr>
</tbody>
</table>

See Also
“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

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
B8601LZw.

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

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

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>21319</td>
<td>153200-0400</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations”
on page 40
B8601TMw.d Format

Writers time values by using the ISO 8601 basic notation \texttt{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= System Option” in SAS System Options: Reference. 

**Supports:** ISO 8601 Element 5.3.1.1, complete representation 

**Syntax**

\texttt{B8601TMw.d} 

**Syntax Description**

\texttt{w} 

specifies the width of the output field. 

Default 8 

Range 6–15 

\texttt{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 \texttt{hhmmss<ffffff>}: 

\texttt{hh} 

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

\texttt{mm} 

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

\texttt{ss} 

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

\texttt{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;

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>55920</td>
<td>153200</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

**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 $\text{hhmmss}+|\text{hhmm}$.

**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

**Syntax**

B8601TXw.

**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:

\( hh\text{mmss}+|\text{hhmm} \)

- **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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data <em>null</em>; t='12:34:56't; put t b8601tx.; run;</td>
<td>073456-0500</td>
</tr>
<tr>
<td>options timezone='America/Adak'; data <em>null</em>; t='12:34:56't; put t b8601tx.; run;</td>
<td>023456-1000</td>
</tr>
</tbody>
</table>

**See Also**

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

---

**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:** 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

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:

- \( hhm\text{ss}\text{+|–}hh:mm \)
  
  Note: Use this form when \( w \) is large enough to support this time notation.

- \( hh\text{m}\text{ss}Z \)
  
  Note: Use this form when \( w \) is not large enough to support the +|–hh:mm time zone notation.

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

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

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

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

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

<table>
<thead>
<tr>
<th>Value for x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>55920</td>
<td>153200+0000</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

**COMMAw.d Format**

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

**Syntax**

`COMMAw.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.

<table>
<thead>
<tr>
<th>Range</th>
<th>0–31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>must be less than ( w )</td>
</tr>
</tbody>
</table>

Details

The \texttt{COMMAw.d} format writes numeric values with a comma that separates every three digits and a period that separates the decimal fraction.

Comparisons

- The \texttt{COMMAw.d} format is similar to the \texttt{COMMAXw.d} format, but the \texttt{COMMAXw.d} format reverses the roles of the decimal point and the comma. This convention is common in European countries.
- The \texttt{COMMAw.d} format is similar to the \texttt{DOLLARw.d} format except that the \texttt{COMMAw.d} format does not print a leading dollar sign.

Example

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

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

<table>
<thead>
<tr>
<th>Value of ( x )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>23451.23</td>
<td>23,451.23</td>
</tr>
<tr>
<td>123451.234</td>
<td>123,451.23</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “\texttt{COMMAXw.d Format}” on page 147
COMMAXw.d Format

Write 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= System Option” in SAS System Options: Reference.

**Syntax**

COMMAXw.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 COMMAXw.d format writes numeric values with a period that separates every three digits and with a comma that separates the decimal fraction.

**Comparisons**

The COMMAw.d format is similar to the COMMAXw.d format, but the COMMAXw.d format reverses the roles of the decimal point and the comma. This convention is common in European countries.

**Example**

```sas
data one;
  input x;
```
data one;
set sashelp.class;
run;

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>23451.23</td>
<td>23.451,23</td>
</tr>
<tr>
<td>123451.234</td>
<td>123.451,23</td>
</tr>
</tbody>
</table>

**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= System Option” in SAS System Options: Reference.

**Syntax**

**Syntax Description**

**w**

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

<table>
<thead>
<tr>
<th>Default</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1–32</td>
</tr>
</tbody>
</table>

**p**

specifies the precision. This argument is optional.

<table>
<thead>
<tr>
<th>Default</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0–16</td>
</tr>
</tbody>
</table>
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 \( Dw.p \) format.

Details

The \( Dw.p \) 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.
- The \( Dw.p \) writes numbers with the desired precision and more alignment than the BEST\( w \) format.
- The BEST\( Dw.p \) format is a combination of the BEST\( w \) format and the \( Dw.p \) 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  
  12.345  
  1.2345  
  .12345
; run;

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>12345.0</td>
</tr>
<tr>
<td>1234.5</td>
<td>1234.5</td>
</tr>
<tr>
<td>Value of x</td>
<td>Result</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>123.45</td>
<td>123.45000</td>
</tr>
<tr>
<td>12.345</td>
<td>12.34500</td>
</tr>
<tr>
<td>1.2345</td>
<td>1.23450</td>
</tr>
<tr>
<td>.12345</td>
<td>0.12345</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “BESTDw.p Format” on page 128

---

**DATEw. Format**

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

**Categories:** CAS

Date and Time

**Alignment:** Right

**Syntax**

`DATEw.`

**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 `DATEw.` 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.
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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x date5.;</td>
<td>15MAR</td>
</tr>
<tr>
<td>put x date6.;</td>
<td>15MAR</td>
</tr>
<tr>
<td>put x date7.;</td>
<td>15MAR18</td>
</tr>
<tr>
<td>put x date8.;</td>
<td>15MAR18</td>
</tr>
<tr>
<td>put x date9.;</td>
<td>15MAR2018</td>
</tr>
<tr>
<td>put x date11.;</td>
<td>15-MAR-2018</td>
</tr>
</tbody>
</table>

See Also

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

Informats:
- “DATEw. Informat” on page 537

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= System Option” in SAS System Options: Reference.

Syntax

DATEAMPM\textsubscript{w.d}

Syntax Description

\textbf{w} \\
specifies the width of the output field. \\
Default 19 \\
Range 7–40 \\
Tip SAS requires a minimum \textit{w} value of 13 to write AM or PM. For widths between 10 and 12, SAS writes a 24-hour clock time.

\textbf{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 \textit{d} must be less than \textit{w}. \\
Note If \textit{w}–\textit{d}< 17, SAS truncates the decimal values.

Details

The DATEAMPM\textsubscript{w.d} format writes SAS datetime values in the form
\textit{ddmmmyy}:\textit{hh}:\textit{mm}:\textit{ss}.\textit{ss}.

\textit{dd} \\
is an integer that represents the day of the month.

\textit{mmm} \\
is the first three letters of the month name.

\textit{yy} \\
is a two-digit integer that represents the year.

\textit{hh} \\
is an integer that represents the hour.

\textit{mm} \\
is an integer that represents the minutes.

\textit{ss}.\textit{ss} \\
is the number of seconds to two decimal places.

Comparisons

The DATEAMPM\textsubscript{w.d} format is similar to the DATETIME\textsubscript{w.d} format, except that DATEAMPM\textsubscript{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;

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x dateampm.;</td>
<td>13MAR18:10:25:33 PM</td>
</tr>
<tr>
<td>put x dateampm7.;</td>
<td>13MAR18</td>
</tr>
<tr>
<td>put x dateampm10.;</td>
<td>13MAR18:22</td>
</tr>
<tr>
<td>put x dateampm13.;</td>
<td>13MAR18:10 PM</td>
</tr>
<tr>
<td>put x dateampm22.2;</td>
<td>13MAR18:10:25:33.00 PM</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DATETIMEw.d Format” on page 153

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= System Option” in SAS System Options:
Reference.

**Syntax**

DATETIMEw.d
### Syntax Description

**w**

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>7–40</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Range</th>
<th>0–39</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>d must be less than w.</td>
</tr>
</tbody>
</table>

### Details

The DATETIMEw.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

```plaintext
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 datetime19.1 ;
  put x datetime20.1 ;
  put x datetime21.2 ;
run;
```
### DAYw. Format

Writes date values as the day of the month.

**Categories:**
- CAS
- Date and Time

**Alignment:**
- Right

**Syntax**

```
DAYw.
```
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;

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x day2. ;</td>
<td>15</td>
</tr>
</tbody>
</table>

DDMMYYw. Format

Writes date values in the form \( ddmm<yy>yy \) or \( dd/mm/yy<y>yy \), where a forward slash is the separator and the year appears as either two or four digits.

Categories: CAS Date and Time

Alignment: Right

Syntax

DDMMYYw.

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 DDMMYYw. format writes SAS date values in the form $ddm<yy>yy$ 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

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x ddmmyy5.;</td>
<td>15/05</td>
</tr>
<tr>
<td>put x ddmmyy6.;</td>
<td>150518</td>
</tr>
<tr>
<td>put x ddmmyy7.;</td>
<td>150518</td>
</tr>
<tr>
<td>put x ddmmyy8.;</td>
<td>15/05/18</td>
</tr>
<tr>
<td>put x ddmmyy10.;</td>
<td>15/05/2018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 150
- “DDMMYYxw. Format” on page 158
- “MMDDYYw. Format” on page 207
- “YYMMDDw. Format” on page 430

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

- “DATEw. Informat” on page 537
- “DDMMYYw. Informat” on page 540
- “MMDDYYw. Informat” on page 565
- “YYMMDDw. Informat” on page 626

**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 and Time

**Alignment:** Right

**Syntax**

`DDMMYY,xw`

**Syntax Description**

- `x` identifies a separator or specifies that no separator appear between the day, the month, and the year. The following 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 DDMMYYxw. format writes SAS date values in the form \( ddm\langle yy\rangle yy \) or \( ddxm\langle yy\rangle yy \):

\( dd \)

is an integer that represents the day of the month.

\( x \)

is a specified separator.

\( mm \)

is an integer that represents the month.

\( \langle yy\rangle yy \)

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;

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x ddmmyyc5.;</td>
<td>15:06</td>
</tr>
<tr>
<td>put x ddmmyyd8.;</td>
<td>15-06-18</td>
</tr>
<tr>
<td>put x ddmmyyp10.;</td>
<td>15.06.2018</td>
</tr>
<tr>
<td>put x ddmmyyn8.;</td>
<td>15062018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 150
- “DDMMYYw. Format” on page 156
- “MMDDYYxw. Format” on page 209
- “YYMMDDXw. Format” on page 432
Functions:
- “DAY Function” in *SAS Functions and CALL Routines: Reference*
- “MDY Function” in *SAS Functions and CALL Routines: Reference*
- “MONTH Function” in *SAS Functions and CALL Routines: Reference*
- “YEAR Function” in *SAS Functions and CALL Routines: Reference*

Informats:
- “DDMMYYw. Informat” on page 540

---

**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.

**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= System Option” 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 DOLLARw.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 DOLLARw.d format is similar to the DOLLARXw.d format, but the DOLLARXw.d format reverses the roles of the decimal point and the comma. This convention is common in European countries.

• The DOLLARw.d format is the same as the COMMAw.d format except that the COMMAw.d format does not write a leading dollar sign.

Example

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

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

See Also

Formats:

• “COMMAw.d Format” on page 145
• “DOLLARXw.d Format” on page 161

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= System Option” in SAS System Options: Reference.

Syntax

DOLLARXw.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 DOLLAR\( X \)\( w \).\( d \) 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 DOLLAR\( X \)\( w \).\( d \) 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\( X \)\( w \).\( d \) format is similar to the DOLLAR\( w \).\( d \) format, but the DOLLAR\( X \)\( w \).\( d \) format reverses the roles of the decimal point and the comma. This convention is common in European countries.

- The DOLLAR\( X \)\( w \).\( d \) 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

```sas
data one;
  netpay=1254.71;
  put netpay dollarx10.2;
run;
```

<table>
<thead>
<tr>
<th>Value of netpay</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1254.71</td>
<td>$1.254,71</td>
</tr>
</tbody>
</table>
See Also

Formats:

- “COMMAXw.d Format” on page 147
- “DOLLARw.d Format” on page 160

**DOWNAMEw. Format**

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

**Categories:** CAS  
Date and Time

**Alignment:** Right

**Syntax**

`DOWNAMEw.`

**Syntax Description**

`w`

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
<th>Tip</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1–32</td>
<td>If you omit <code>w</code>, SAS prints the entire name of the day.</td>
</tr>
</tbody>
</table>

**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**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate downame.;</td>
<td>Wednesday</td>
</tr>
</tbody>
</table>

---
See Also

Formats:

- “WEEKDAYw. Format” on page 416

**DTDATEw. Format**

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

- **Category:** Date and Time
- **Alignment:** Right
- **Restriction:** This format is not supported in a DATA step that runs in CAS.

**Syntax**

`DTDATEw.`

**Syntax Description**

`w`

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>5–9</td>
</tr>
<tr>
<td>Tip</td>
<td>Use a width of 9 to print a four-digit year.</td>
</tr>
</tbody>
</table>

**Details**

The DTDATEw. 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 DTDATEw. format produces the same type of output that the DATEw. format produces. The difference is that the DTDATEw. format requires a datetime value.

**Example**

```sas
data one;
  mydt='28mar2018 13:45:14'dt;
  put mydt ddate.;
  put mydt ddate9.;
run;
```
### SAS Statement

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt ddate.;</td>
<td>28MAR18</td>
</tr>
<tr>
<td>put mydt ddate9.;</td>
<td>28MAR2018</td>
</tr>
</tbody>
</table>

---

### See Also

 Formats:
- “DATEw. Format” on page 150

---

### DTMONYYw. Format

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

**Categories:**
- CAS
- Date and Time

**Alignment:** Right

---

### Syntax

**DTMONYYw.**

**Syntax Description**

`w`

specifies the width of the output field.

**Default**

5

**Range**

5–7

---

### Details

The `DTMONYYw. 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 `DTMONYYw. format` and the `MONYYw. format` are similar in that they both write date values. The difference is that `DTMONYYw. format` expects a datetime value as input, and `MONYYw. format` 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;

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt dtmonyy.;</td>
<td>MAR18</td>
</tr>
<tr>
<td>put mydt dtmonyy5.;</td>
<td>MAR18</td>
</tr>
<tr>
<td>put mydt dtmonyy6.;</td>
<td>MAR18</td>
</tr>
<tr>
<td>put mydt dtmonyy7.;</td>
<td>MAR2018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATETIMEw.d Format” on page 153
- “MONYYw. Format” on page 218

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 and Time
Alignment: Right

Syntax

DTWKDATXw.

Syntax Description

w
  specifies the width of the output field.

Default  29
Range    3–37
Details

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

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt dtwkdatx.;</td>
<td>Wednesday, 28 March 2018</td>
</tr>
<tr>
<td>put mydt dtwkdatx3.;</td>
<td>Wed</td>
</tr>
<tr>
<td>put mydt dtwkdatx8.;</td>
<td>Wed</td>
</tr>
<tr>
<td>put date dtwkdatx25.;</td>
<td>Wednesday, 28 Mar 2018</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DATETIMEw.d Format” on page 153
- “WEEKDATXw. Format” on page 414
DTYEARw. Format

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

Categories:
CAS
Date and Time

Alignment: Right

Syntax

\texttt{DTYEARw.}

Syntax Description

\texttt{w}

specifies the width of the output field.

Default \ 4

Range \ 2–4

Details

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

Example

\begin{verbatim}
data one;
  mydt='28mar2018 13:45:14'dt;
  put mydt dtyear. ;
  put mydt dtyear2. ;
  put mydt dtyear3. ;
  put mydt dtyear4. ;
run;
\end{verbatim}

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt dtyear. ;</td>
<td>2018</td>
</tr>
<tr>
<td>put mydt dtyear2. ;</td>
<td>18</td>
</tr>
<tr>
<td>put mydt dtyear3. ;</td>
<td>18</td>
</tr>
<tr>
<td>put mydt year4. ;</td>
<td>2018</td>
</tr>
</tbody>
</table>
See Also

Formats:
- “DATETIMEw.d Format” on page 153
- “YEARw. Format” on page 427

DTYYQcw. Format

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

**Categories:** CAS

**Alignment:** Right

**Syntax**

```
DTYYQcw.
```

**Syntax Description**

`w`

specifies the width of the output field.

**Default** 4

**Range** 4–6

**Details**

The DTYYQcw. 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**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt dtyyqc.;</td>
<td>18:1</td>
</tr>
<tr>
<td>put mydt dtyyqc4.;</td>
<td>18:1</td>
</tr>
</tbody>
</table>

---
Ew. Format

Writes numeric values in scientific notation.

Categories: CAS
Numeric

Alignment: Right

Syntax

\texttt{Ew.}

Syntax Description

\texttt{w}

\texttt{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

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

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>1.257E+03</td>
</tr>
<tr>
<td>1257</td>
<td>1.257E+03</td>
</tr>
<tr>
<td>-1257</td>
<td>-1.257E+03</td>
</tr>
</tbody>
</table>

**E8601DAw. Format**

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

**Categories:** CAS
- Date and Time
- ISO 8601

**Alignment:** Left

**Alias:** IS8601DAw.

**Restriction:** UTC time zone offset values are not supported.

**Supports:** ISO 8601 Element 5.2.1.1, complete representation

**Syntax**

E8601DAw.

**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;

<table>
<thead>
<tr>
<th>Value for mydt</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>21442</td>
<td>2018-09-15</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

E8601DNw. Format

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

Categories: CAS
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

E8601DNw.

Syntax Description

w

specifies the width of the input field.

Default 10

Requirement The width of the input field must be 10.

Details

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

yyyy

is a four-digit year.

mm

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

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

**Example**

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

<table>
<thead>
<tr>
<th>Value for mydt</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1852642342</td>
<td>2018-09-15</td>
</tr>
</tbody>
</table>

**See Also**

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

---

**E8601DTw.d Format**

Writes datetime values by using the ISO 8601 extended notation \( yyyy-mm-ddThh:mm:ss.ffffff \).

**Categories:** CAS
  Date and Time
  ISO 8601

**Alignment:** Left

**Alias:** IS8601DTw.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= System Option” in SAS System Options: Reference.

**Supports:** ISO 8601 Element 5.4.1, complete representation

**Syntax**

\[ \text{E8601DTw.d} \]

**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 \( yyyy-mm-ddThh:mm:ss.ffffff \):

- \( yyyy \)
  - 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.
- \( fffffff \)
  - 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**

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

<table>
<thead>
<tr>
<th>Value of mydt</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1852642342</td>
<td>2018-09-15T14:52:22.000</td>
</tr>
</tbody>
</table>

**See Also**

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40
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 \textit{yyyy-mm-ddThh:mm:ss+hh:mm}.

\begin{itemize}
  \item \textbf{Categories:} CAS
  \item \textbf{Date and Time}
  \item \textbf{ISO 8601}
  \item \textbf{Alignment:} Left
  \item \textbf{Supports:} ISO 8601 Element 5.4.1, complete representation
\end{itemize}

\section*{Syntax}

\texttt{E8601DXw}.

\section*{Syntax Description}

\textit{w}

\begin{itemize}
  \item specifies the width of the output field.
  \item \textbf{Default} 26
  \item \textbf{Range} 20–35
\end{itemize}

\section*{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 \texttt{TIMEZONE=} system option. If the \texttt{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:

\begin{itemize}
  \item \textit{yyyy-mm-ddThh:mm:ss+hh:mm}
\end{itemize}

- \textit{yyyy} is a four-digit year.
- \textit{mm} is a two-digit month (zero padded) between 01 and 12.
- \textit{dd} is a two-digit day of the month (zero padded) between 01 and 31.
- \textit{hh} is a two-digit hour (zero padded) between 00 and 23.
- \textit{mm} is a two-digit minute (zero padded) between 00 and 59.
- \textit{ss} is a two-digit second (zero padded) between 00 and 59.
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data <em>null</em>; t='15Sep2108T12:34:56'dt ; put t e8601dx.; run;</td>
<td>2018-09-15T08:34:56-04:00</td>
</tr>
<tr>
<td>options timezone='America/Adak'; data <em>null</em> ; t='15Sep2108T12:34:56'dt ;</td>
<td>2018-09-15T03:34:56-09:00</td>
</tr>
<tr>
<td>put t e8601dx.; run;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

**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-ddThh:mm:ss+00:00.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Date and Time</td>
</tr>
<tr>
<td></td>
<td>ISO 8601</td>
</tr>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
<tr>
<td>Alias:</td>
<td>IS8601DZw.</td>
</tr>
<tr>
<td>Restriction:</td>
<td>UTC time zone offset values are not supported.</td>
</tr>
<tr>
<td>Supports:</td>
<td>ISO 8601 Element 5.4.1, complete representation</td>
</tr>
</tbody>
</table>

**Syntax**

E8601DZw.
**Syntax Description**

 substituted for by 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-ddThh:mm:ss+00:00
  
  *Note:* Use this form when w is large enough to support this time zone notation.

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

**Example**

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

<table>
<thead>
<tr>
<th>Value of mydt</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1852642342</td>
<td>2018-09-15T14:52:22+00:00</td>
</tr>
</tbody>
</table>

### See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

---

**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-ddThh:mm:ss+|–hh:mm`.

**Categories:**
- CAS
- Date and Time
- ISO 8601

**Alignment:** Right

**Supports:** ISO 8601 Elements 5.3.3 and 5.3.4.2

---

**Syntax**

`E8601LXw`

**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-ddThh: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.
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:

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

<table>
<thead>
<tr>
<th>Value of mydt</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1852634096</td>
<td>2018-09-15T12:34:56-05:00</td>
</tr>
</tbody>
</table>

### See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

---

**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  
Date and Time  
ISO 8601

**Alignment:** Left

**Alias:** IS8601LZw.

**Supports:** ISO 8601 Element 5.3.1.1, complete representation

**Syntax**

E8601LZw.
**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 \( hh:mm\.fffff \), 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:

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

<table>
<thead>
<tr>
<th>Value of mytime</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>53546</td>
<td>14:52:26-05:00</td>
</tr>
</tbody>
</table>

**See Also**

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

---

**E8601TMw.d Format**

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

**Categories:** CAS  
Date and Time  
ISO 8601

**Alignment:** Left  

**Alias:** IS8601TMw.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= System Option” 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**

E8601TMw.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.
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**

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

<table>
<thead>
<tr>
<th>Value of mytime</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>53546</td>
<td>14:52:26</td>
</tr>
</tbody>
</table>

**See Also**

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

---

**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 \( hh:mm:ss\pm mm \).

**Categories:** CAS, Date and Time, ISO 8601

**Alignment:** Right

**Supports:** ISO 8601 Elements 5.3.3 and 5.3.4

**Syntax**

E8601TXw.
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>data <em>null</em> ;</td>
<td></td>
</tr>
<tr>
<td>t='12:34:56't;</td>
<td>07:34:56-05:00</td>
</tr>
<tr>
<td>put t e8601tx.;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>
Statement | Result
--- | ---
```options timezone='America/Adak';
data _null_;
t='12:34:56't;
put t e8601tx.;
run;```
| 02:34:56-10:00

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

**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
Date and Time
ISO 8601

**Alignment:** Left

**Alias:** IS8601TZw.d

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

**Supports:** ISO 8601 Element 5.3.1.1, complete representation

**Syntax**

`E8601TZw.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
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.

- \( hh:mm:ss\)\( Z \)
  
  **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.

\( .fff \)

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**

```sas
data _null_;
  input @1 time e8601tz.;
  put time=;
  put time e8601tz.;
datalines;
```
run;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value of time</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put time=;</td>
<td>17024</td>
<td>04:43:44+00:00</td>
</tr>
<tr>
<td>put time e8601tz.;</td>
<td>85424</td>
<td>23:43:44+00:00</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 40

**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**

```
EUROw.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 EUROw.d format is similar to the EUROXw.d format, but EUROXw.d format reverses the roles of the decimal point and the comma. This convention is common in European countries.

- The EUROw.d format is similar to the DOLLARw.d format, except that DOLLARw.d format writes a leading dollar sign instead of the euro symbol.

Example

These examples use 1254.71 as the value of amount.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put amount euro10.2;</td>
<td>B1,254.71</td>
</tr>
<tr>
<td>put amount euro5.;</td>
<td>1,255</td>
</tr>
<tr>
<td>put amount euro9.2;</td>
<td>B1,254.71</td>
</tr>
<tr>
<td>put amount euro15.3;</td>
<td>B1,254.710</td>
</tr>
</tbody>
</table>

```sas
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:  
B1,254.71  
1,255  
B1,254.71  
B1,254.710  
/
/* This code determines the default length. */

data _null_;  
input x;  
put x euro.;  
datalines;  
1  
22  
333  
4444  
55555  
666666  
7777777  
88888888  
999999999  
```
/* 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 Format” on page 189

Informats:
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**

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

**Example**

These examples use 1254.71 as the value of amount.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put amount eurox10.2;</td>
<td>B1.254,71</td>
</tr>
</tbody>
</table>
### Statements
<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put amount eurox5.;</td>
<td>1.255</td>
</tr>
<tr>
<td>put amount eurox9.2;</td>
<td>E1.254,71</td>
</tr>
<tr>
<td>put amount eurox15.3;</td>
<td>E1.254,710</td>
</tr>
</tbody>
</table>

```sas
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. */
```sas
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  
S5.555  
6666666  
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 Format” on page 186

Informats:


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

**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.

**Comparisons**

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

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

**Example**

```plaintext
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.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Value of y</th>
</tr>
</thead>
<tbody>
<tr>
<td>put y=;</td>
<td>€?</td>
</tr>
<tr>
<td>put y=hex.;</td>
<td>0000803F</td>
</tr>
</tbody>
</table>

**FRACTw. Format**

Converts numeric values to fractions.

- **Categories:** CAS
  - Numeric
- **Alignment:** Right
Syntax

FRACTw.

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 FRACTw. format. FRACTw. writes fractions in reduced form (for example, 1/2 instead of 50/100).

Example

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6666666667</td>
<td>2/3</td>
</tr>
<tr>
<td>0.2784</td>
<td>174/625</td>
</tr>
</tbody>
</table>

HEXw. Format

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

Categories: CAS Numeric

Alignment: Left

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

HEXw.

Syntax Description

w

specifies the width of the output field.

Default 8

Range 1–16

Tip If w<16, the HEXw 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, HEXw displays floating-point values in their hexadecimal form.

Details

In any operating environment, the least significant byte written by HEXw is the rightmost byte. Some operating environments store integers with the least significant digit as the first byte. The HEXw 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 HEXw numeric format and the $HEXw character format both generate the hexadecimal equivalent of values.

Example

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

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>194</td>
<td>194</td>
</tr>
<tr>
<td>-150</td>
<td>-150</td>
</tr>
</tbody>
</table>
### HHMMw.d Format

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

**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= System Option" in SAS System Options: Reference.

---

#### Syntax

**HHMMw.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 HHMMw.d format writes SAS time values in the form *hh:mm*: 

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>35.4</td>
<td>00000023</td>
</tr>
<tr>
<td>88</td>
<td>00000058</td>
</tr>
<tr>
<td>2.33</td>
<td>0000002</td>
</tr>
<tr>
<td>-150</td>
<td>FFFFFFF6A</td>
</tr>
</tbody>
</table>
hh

is an integer.

*Note:* If hh is a single digit, HHMMw.d places a leading blank before the digit. For example, the HHMMw.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 HHMMw.d format is similar to the TIMEw.d format except that the HHMMw.d format does not print seconds.

The HHMMw.d format writes a leading blank for a single-hour digit. The TODw.d format writes a leading zero for a single-hour digit.

**Example**

```sas
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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mytime hhmm.;</td>
<td>13:00</td>
</tr>
<tr>
<td>put mytime hhmm8.2;</td>
<td>12:59.93</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “HOURw.d Format” on page 197
- “MMSSw.d Format” on page 211
- “TIMEw.d Format” on page 402
- “TODw.d Format” on page 406

**Functions:**

- “HMS Function” in *SAS Functions and CALL Routines: Reference*
- “HOUR Function” in *SAS Functions and CALL Routines: Reference*
- “MINUTE Function” in *SAS Functions and CALL Routines: Reference*
Informats:

- “TIMEw. Informat” on page 610

---

**HOURw.d Format**

Writs time values as hours and decimal fractions of hours.

**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= System Option” in *SAS System Options: Reference*.

**Syntax**

**HOURw.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**

```sas
data a;
    mytime='12:59:56't;
    put mytime hour.;
```

run;

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mytime hour;</td>
<td>13</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “HHMMw.d Format” on page 195
- “MMSSw.d Format” on page 211
- “TIMEw.d Format” on page 402
- “TODw.d Format” on page 406

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

**Informats:**
- “TIMEw. Informat” on page 610

**IBw.d Format**

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

<table>
<thead>
<tr>
<th>Category:</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
<tr>
<td>Restriction:</td>
<td>This format is not supported in a DATA step that runs in CAS.</td>
</tr>
</tbody>
</table>

**See:**
- “IB Format: UNIX” in *SAS Companion for UNIX Environments*
- “IBw.d Format: Windows” in *SAS Companion for Windows*
- “IBw.d Format: z/OS” in *SAS Companion for z/OS*

**Syntax**

`IBw.d`

**Syntax Description**

`w`

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

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

Default 0
Range 0–10

Details
The IBw.d format writes integer binary (fixed-point) values, including negative values that are represented in two’s-complement notation. IBw.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 33.

Comparisons
The IBw.d and PIBw.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 IBRw.d and PIBRw.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 33.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

Example

data _null_;  
x=128;  
y=put(x,ib4.);  
put y $hex8.;  
run;

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result on Big Endian Platforms</th>
<th>Result on Little Endian Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>00000080</td>
<td>8000000</td>
</tr>
</tbody>
</table>

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

See Also

Formats:
IBRw.d Format

 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**

IBRw.d

**Syntax Description**

- **w**
  - Specifies the width of the output field.
  - **Default:** 4
  - **Range:** 1–8

- **d**
  - Specifies to multiply the number by 10d. This argument is optional.
  - **Default:** 0
  - **Range:** 0–10

**Details**

The IBRw.d format writes integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. IBRw.d writes integer binary values that are generated by and for Intel and DEC operating environments. Use IBRw.d to write integer binary data from Intel or DEC environments on other operating environments. The IBRw.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 33.

**Comparisons**

- The IBw.d and PIBw.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 IBRw.d and PIBRw.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 IBw.d and IBRw.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 33.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

**Example**

```sas
data _null_;
x=128;
y=put(x,ibr4.);
put y $hex8.;
run;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>80000000</td>
</tr>
</tbody>
</table>

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

**See Also**

**Formats:**

- “IBw.d Format” on page 198

---

**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**

**IEEEw.d**

**Syntax Description**

\( w \)

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1–8</td>
</tr>
</tbody>
</table>
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

```plaintext
data _null_;
    x=1;
    test1=put(x,ieee4.);
    put test1 $hex8.;
    test2=put(x,ieee5.);
    put test2 $hex10.;
run;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3F800000</td>
</tr>
<tr>
<td>3F000000</td>
<td></td>
</tr>
</tbody>
</table>

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

JULDAY\( w \). Format

Writes date values as the Julian day of the year.

- **Category:** Date and Time
- **Alignment:** Right
- **Restriction:** This format is not supported in a DATA step that runs in CAS.
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 \textit{ddd}. In this syntax, the letter \textit{d} is defined as follows:

\begin{itemize}
  \item \textit{ddd}\ is the number of the day, 1–365 (or 1–366 for leap years).
\end{itemize}

Example

```sas
data one;
input date :mmddyy10.;
datalines;
01/12/2018
07/04/2018
;
r
run;

data two;
set one;
put date julday3.;
r
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.

<table>
<thead>
<tr>
<th>Input Value</th>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>21196</td>
<td>put date julday3.;</td>
<td>12</td>
</tr>
<tr>
<td>21369</td>
<td>put date julday3.;</td>
<td>185</td>
</tr>
</tbody>
</table>
**JULIANw. Format**

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

**Categories:** CAS

Date and Time

**Alignment:** Left

**See:** "Julian Date Formats and Astronomical Dates" on page 9

---

**Syntax**

`JULIANw.`

**Syntax Description**

`w` specifies the width of the output field.

- **Default:** 5
- **Range:** 5–7

**Tip**

If `w` is 5, the JULIANw. format writes the date with a two-digit year. If `w` is 7, the JULIANw. format writes the date with a four-digit year.

---

**Details**

The JULIANw. 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**

```sas
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).
### MDYAMPw.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= System Option" in SAS System Options: Reference.

**Note:** The default time period is AM.

### Syntax

**MDYAMPw.**

### 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
Details
The MDYAMPMw.d format writes SAS datetime values in the following form:

\[ \text{mm/dd/yy<yy> hh:mm<AM | PM>} \]

- \( \text{mm} \) is an integer between 1 and 12 that represents the month.
- \( \text{dd} \) is an integer between 1 and 31 that represents the day of the month.
- \( \text{yy} \) or \( \text{yyyy} \) specifies a two-digit or four-digit integer that represents the year.
- \( \text{hh} \) is an integer between 00 and 23 that represents hours.
- \( \text{mm} \) is an integer between 00 and 59 that represents minutes.
- \( \text{AM | PM} \) 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.
- \( \text{date and time separator characters} \) 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 MDYAMPMw. format writes datetime values with separators in the form \( \text{mm/dd/yy<yy> hh:mm AM | PM} \), and requires a space between the date and the time.

The DATETIMEw.d format writes datetime values with separators in the form \( \text{ddmmmyy<yy>: hh:mm:ss.ss} \).

Example
```
data one;
  mydt='04jul2018 14:58:23'dt;
  put mydt mdyampm25.;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt mdyampm25.</td>
<td>7/4/2018 2:58 PM</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DATETIMEw.d Format” on page 153
• “MDYAMPw.d Informat” on page 563

### 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 and Time  
**Alignment:** Right

#### Syntax

`MMDDYYw.`

#### 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 MMDDYYw. 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

```sas
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;

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt=02may2018’d;</td>
<td>05/02/18</td>
</tr>
<tr>
<td>put mydt mmddyy2.;</td>
<td>05</td>
</tr>
<tr>
<td>put mydt mmddyy3.;</td>
<td>05</td>
</tr>
<tr>
<td>put mydt mmddyy4.;</td>
<td>0502</td>
</tr>
<tr>
<td>put mydt mmddyy5.;</td>
<td>05/02</td>
</tr>
<tr>
<td>put mydt mmddyy6.;</td>
<td>050218</td>
</tr>
<tr>
<td>put mydt mmddyy7.;</td>
<td>050218</td>
</tr>
<tr>
<td>put mydt mmddyy8.;</td>
<td>05/02/18</td>
</tr>
<tr>
<td>put mydt mmddyy10.;</td>
<td>05/02/2018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 150
- “DDMMYYw. Format” on page 156
- “MMDDYYxw. Format” on page 209
- “YYMMDDw. Format” on page 430

Functions:
- “DAY Function” in SAS Functions and CALL Routines: Reference
- “MDY Function” in SAS Functions and CALL Routines: Reference
- “MONTH Function” in SAS Functions and CALL Routines: Reference
- “YEAR Function” in SAS Functions and CALL Routines: Reference

Informats:
- “DATEw. Informat” on page 537
MMDDYYxw. Format

 Writes date values in the form `mmdd<yy>` or `mm-dd<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.

**Syntax**

`MMDDYYxw`.

**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**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>7</td>
</tr>
</tbody>
</table>

**Range**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>5–32</td>
</tr>
</tbody>
</table>

**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 MMDDYYxw. 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

```sas
data one;
  mydt='02may2018'd;
  put mydt mmdyyc5.;
  put mydt mmdyyd8.;
  put mydt mmdyyp10.;
  put mydt mmdyyyn8.;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt mmdyyc5.;</td>
<td>05:02</td>
</tr>
<tr>
<td>put mydt mmdyyd8.;</td>
<td>05-02-18</td>
</tr>
<tr>
<td>put mydt mmdyyp10.;</td>
<td>05.02.2018</td>
</tr>
<tr>
<td>put mydt mmdyyyn8.;</td>
<td>05022018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 150
- “DDMMYYxw. Format” on page 158
- “MMDDYYw. Format” on page 207
- “YYMMDxDDw. Format” on page 432

Functions:
- “DAY Function” in SAS Functions and CALL Routines: Reference
Informats:

- “MMDDYYw. Informat” on page 565

### MMSS\(w.d\) Format

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

**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= System Option” in SAS System Options: Reference.

**Syntax**

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

**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**

```sas
data a;
  x='0:30:21't;
  y='25:42:08't;
  put x mmss8.;
```

```
put y mmss10.2;
run;
```

### SAS Statement Result

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put y mmss10.2;</td>
<td>1542:08.00</td>
</tr>
<tr>
<td>put x mmss8.;</td>
<td>510:21</td>
</tr>
<tr>
<td>put y mmss10.2;</td>
<td>1542:08.00</td>
</tr>
</tbody>
</table>

### See Also

**Formats:**
- “HHMMw.d Format” on page 195
- “TIMEw.d Format” on page 402

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

**Informats:**
- “TIMEw. Informat” on page 610

---

### 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.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Date and Time</td>
</tr>
</tbody>
</table>

| Alignment:        | Right      |

### Syntax

**MMYYw.**

### Syntax Description

`w`

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>5–32</td>
</tr>
</tbody>
</table>
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 MMYYw. format writes SAS date values in the form \( mmM<yy>yy \). 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

```sas
data one;
  dte='15may2018'd;
  put dte mmyy.;
  put dte mmyy5.;
  put dte mmyy6.;
  put dte mmyy7.;
  put dte mmyy10.;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dte mmyy.</td>
<td>05M2018</td>
</tr>
<tr>
<td>put dte mmyy5.</td>
<td>05M18</td>
</tr>
<tr>
<td>put dte mmyy6.</td>
<td>05M18</td>
</tr>
<tr>
<td>put dte mmyy7.</td>
<td>05M2018</td>
</tr>
<tr>
<td>put date mmyy10.</td>
<td>05M2018</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “MMYYXw. Format” on page 214
- “YYMMw. Format” on page 429
**MMYYwx. 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
Date and Time

**Alignment:** Right

---

**Syntax**

**MMYYwx.**

**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 MMYYyw. format writes SAS date values in one of these forms:

\[ mn<yy>yy \]
\[ mnx<yy>yy \]

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

```sas
data one;
  dte='15may2018'd;
  put dte mmyyc5.;
  put dte mmyyd.;
  put dte mmyyn4.;
  put dte mmyyp8.;
  put dte mmyys10.;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dte mmyyc5.;</td>
<td>05:18</td>
</tr>
<tr>
<td>put dte mmyyd.;</td>
<td>05-2018</td>
</tr>
<tr>
<td>put dte mmyyn4.;</td>
<td>0518</td>
</tr>
<tr>
<td>put dte mmyyp8.;</td>
<td>05.2018</td>
</tr>
<tr>
<td>put dte mmyys10.;</td>
<td>05/2018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “MMYYw. Format” on page 212
- “YYMMxw. Format” on page 434
MONNAMEw. Format

Writes date values as the name of the month.

**Categories:** CAS
Date and Time

**Alignment:** Right

### Syntax

```
MONNAMEw.
```

### 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;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dte monname.;</td>
<td>May</td>
</tr>
<tr>
<td>put dte monname1.;</td>
<td>M</td>
</tr>
<tr>
<td>put dte monname3.;</td>
<td>May</td>
</tr>
</tbody>
</table>

### See Also

**Formats:**
MONTHw. Format

Writes date values as the month of the year.

Categories:
- CAS
  - Date and Time
Alignment: Right

Syntax

MONTHw.

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

Example

```sas
data one;
  dte='15may2018'd;
  put dte month.;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dte month.;</td>
<td>5</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “MONNAMEw. Format” on page 216
MONYYw. Format

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

Categories: CAS
            Date and Time

Alignment: Right

Syntax

MONYYw.

Syntax Description

w
    specifies the width of the output field.

Default  5
Range     5–7

Details

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

Example

   data one;
      dte='15may2018'd;
      put dte monyy5.;
      put dte monyy7.;
   run;

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>----+-----1</td>
</tr>
<tr>
<td>put dte monyy5.;</td>
<td>MAY18</td>
</tr>
</tbody>
</table>
SAS Statement | Result
---|---
put dte monyy7.; | MAY2018

See Also

Formats:
- “DDMMYYw. Format” on page 156
- “DTMONYYw. Format” on page 165
- “MMDDYYw. Format” on page 207
- “YYMMDDw. Format” on page 430

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

Informat:
- “MONYYw. Informat” on page 567

**NEGPARENw.d Format**

Writes negative numeric values in parentheses.

- **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= System Option” in *SAS System Options: Reference*.

**Syntax**

```sas
NEGPARENw.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 `NEGPARENw.d` format attempts to right-align output values. If the input value is negative, `NEGPARENw.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, `NEGPARENw.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 `NEGPARENw.d` format is similar to the `COMMAw.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;

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>1000</td>
<td>1,000</td>
</tr>
<tr>
<td>-200</td>
<td>(200)</td>
</tr>
<tr>
<td>-2000</td>
<td>(2,000)</td>
</tr>
</tbody>
</table>
NENGOw. Format

Writs date values as Japanese dates in the form e.yymmdd.

**Category:** Date and Time  
**Alignment:** Left  
**Restriction:** This format is not supported in a DATA step that runs in CAS.

**Syntax**

NENGOw

**Syntax Description**

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

**Details**

The NENGOw. 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.

```sas
data _null_;  
  date=15342;  
  put date nengo3.;  
  put date nengo6.;  
  put date nengo8.;  
  put date nengo9.;  
  put date nengo10.;  
run;
```
See Also

Informat:

NLBESTw. Format

Writers the best numerical notation based on the locale.

**Categories:** CAS
- Numeric

**Alignment:** Right

**Syntax**

NLBESTw:

**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 Informat: Reference.

**Example**

The following code produces results based on the locale:

```sas
options locale=English_UnitedStates;
data test;
x=-1257000;
put x nlbest6.;
put x nlbest3.;
```

222 Chapter 2 • Dictionary of Formats
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;

options locale=ar_BH;

  -126E4
  ***
  =====
  -0.1
  -,1
  =====
  0.1
  0,1
  =====
  1.26E6
  1E6
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**

Converts a SAS date value to the date value of the specified locale, and then writes the date value as a date.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**

NLDATEw:

**Syntax Description**

w

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>10–200</td>
</tr>
</tbody>
</table>
Comparisons

NLDATEw. is similar to DATEw. and WORDDATEw. except that NLDATEw. 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. Format” on page 231
- “NLDATEWw. Format” on page 233
- “NLDATEWNw. Format” on page 234

### NLDATELw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in the form month, date, year.

**Categories:** CAS
Date and Time

**Alignment:** Left

### Syntax

NLDATELw.
**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.

```sas
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=18 max=200 ----+
November 19, 2012
11/19/2012
Nov 19, 2012
November 19, 2012
November 19, 2012
```

**NLDATEmw. Format**

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date in a medium-uniform pattern.

**Categories:** CAS  
Date and Time

**Alignment:** Left

**Syntax**

```
NLDATEmw
```
Syntax Description

\[ w \]

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

<table>
<thead>
<tr>
<th>Default</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>2–200</td>
</tr>
</tbody>
</table>

Details

NLDATEM writes the date in a medium-uniform pattern such as Nov 19, 2012.

Example

This example specifies the date Nov 19, 2012.

```sas
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

Converts the SAS date value to the date value of the specified locale, and then writes the value as the name of the month and the day of the month.

**Categories:** CAS

Date and Time

**Alignment:** Left

Syntax

NLDATEMDw.

Syntax Description

\[ w \]

specifies the width of the output field.
Example

This example specifies the english_united_states locale option.

```sas
option locale=en_US;
data _null_;  	dy = date();  
put dy nldatemd.;  
run;
```

April 03

See Also

Format:

- “NLDATEGw. Format” on page 236

NLDATEGw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.

- **Categories:** CAS
  - Date and Time
- **Alignment:** Left

**Syntax**

NLDATEGw.

**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**

NLDATEGDL 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.

```sas
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 nldatemdl15.;  
put dy nldatemdl19.;  
put dy nldatemdl12.;  
put dy nldatemdl200.;  
run;
```

```
+--- NLDATEMDL min=5 default=12 max=200 ---+
November 19
11/19
Nov 19
November 19
November 19
```

**NLDATEMDMw. Format**

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.

**Categories:** CAS

**Date and Time**

**Alignment:** Left

**Syntax**

NLDATEMDMw.

**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.

```sas
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;
```

---
Nov 19
11/19
Nov 19
Nov 19
---

NLDATEMDSw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month.

**Categories:** CAS
                  Date and Time

**Alignment:** Left

**Syntax**

NLDATEMDSw.

**Syntax Description**

`w`

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5–200</td>
</tr>
</tbody>
</table>

**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=5 max=200 ----+
11/19
11/19
11/19
11/19

-- NLDATEMNw. Format --

Converts a SAS date value to the date value of the specified locale, and then writes the value as the name of the month.

**Categories:** CAS

Date and Time

**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

**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.

```sas
option locale=en_US;
```
February

option locale=german_germany;
data _null_;  
    month=15760;  
    put month nldatemn.;  
run;

Februar

See Also

Formats:
- “NLDATEx. Format” on page 224
- “NLDATExW. Format” on page 233
- “NLDATENxW. Format” on page 234

NLDATExw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date string.

Categories: CAS
Date and Time

Alignment: Left

Syntax

NLDATExw.

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.

```sas
option locale=en_US;
data _null_
  dt = dt='19Nov2012:00:00:00'dt;
  dy='19Nov2012'd;
  put '----- NLDATEMDL min=2 default=10 max=200 ----'
     +--- NLDATEMDL min=2 default=10 max=200 ---+
       11/19/2012
       11/19/12
       11/19/2012
       11/19/2012
       11/19/2012
     run;
```

NLDATEWw. Format

Converts a SAS date value to the date value of the specified locale, and then writes the value as the date and the day of the week.

Categories: CAS
            Date and Time

Alignment: Left

Syntax

NLDATEWw.

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.
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.

```sas
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
```

```sas
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:

- “NLDATE\(w\). Format” on page 224
- “NLDatem\(n\)w. Format” on page 231
- “NLDATE\(w\)MN\(n\)w. Format” on page 234

**NLDATE\(w\)MN\(n\)w. Format**

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the day of the week.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Date and Time</td>
</tr>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
</tbody>
</table>
Syntax

\texttt{NLDATENWw.}

\textbf{Syntax Description}

\texttt{w}

specifies the width of the output field. If necessary, SAS abbreviates the day of the week to fit the format width.

\begin{itemize}
  \item Default: 9
  \item Range: 4–200
\end{itemize}

\textbf{Comparisons}

\texttt{NLDATENWw.} is similar to \texttt{DOWNAMEw.} except that \texttt{NLDATENWw.} is locale-specific.

\textbf{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 \texttt{en_US} locale. The second example specifies the \texttt{de_DE} locale.

```sas
options locale=en_US;
data _null_;  
  dy=15760;  
  put dy nldatewn.;  
run;
```

\begin{center}
\textbf{Monday}
\end{center}

```sas
options locale=de_DE;
data _null_;  
  dy=15760;  
  put dy nldatewn10.;  
run;
```

\begin{center}
\textbf{Montag}
\end{center}

\textbf{See Also}

\textbf{Formats:}

- “\texttt{NLDATENw. Format}” on page 224
- “\texttt{NLDATENMNw. Format}” on page 231
- “\texttt{NLDATENWw. Format}” on page 233
NLDATEYMW. Format

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the name of the month.

**Categories:**
- CAS
- Date and Time

**Alignment:** Left

**Syntax**

NLDATEYMWw.

**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:

```sas
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;
```

**Example**

This example specifies the spanish_Spain locale option. This example specifies the date of April 4, 2019.

```sas
options locale=spanish_Spain;
data _null_
  dy=today();
  put dy nldateym.;
  put dy nldateym12.;
  put dy nldateym200.;
```
See Also

Format:

• “NLDATEMDw. Format” on page 227

NLDATEYMLw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the month and year.

Categories: CAS
Date and Time

Alignment: Left

Syntax

NLDATEYMLw.

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.

```sas
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.;
run;
```
NLDATEYMMw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date values as the month and year with abbreviations.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**

`NLDATEYMMw.`

**Syntax Description**

`w`

specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>5–200</td>
</tr>
</tbody>
</table>

**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.

```sas
options locale=en_US;
data _null_;  
dt = datetime();  
dy = date();  
put "+--- NLDATEYML min=5 default=14 max=200 ----+";  
put dy nldateyml200.;  
run;
```

---

```
+--- NLDATEYML min=5 default=14 max=200 ----+
April 2019
04/2019
Apr 2019
April 2019
April 2019
```

---
**NLDATEYMSw. Format**

Converts a SAS date value to the date string of the specified locale, and then writes the date value as a date and year.

**Categories:**
- CAS
- Date and Time

**Alignment:** Left

**Syntax**

`NLDATEYMSw.`

**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/yyyy.

**Example**

This example specifies the date April 4, 2019. This example specifies the en_US locale.

```sas
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;
```
NLDATEYQw. Format

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the quarter.

**Categories:**
CAS
Date and Time

**Alignment:**
Left

**Syntax**

NLDATEYQw.

**Syntax Description**

`w`

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>4–200</td>
</tr>
</tbody>
</table>

**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.

```plaintext
options locale=fr_FR;
data _null_;   dy=today();   dt=datetime();   put "---- NLDATEYQ min=4 default=16 max=200 ----";   put '16' +5 dy nldateyq.;   put '14' +5 dy nldateyq14.;   put '32' +5 dy nldateyq14.;   put '200' +5 dy nldateyq200.;run;
```
**NLDATEYQLw. Format**

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year’s quarter value (Q1–Q4) using abbreviations.

**Categories:** CAS  
Date and Time  

**Alignment:** Left

**Syntax**

\[ \text{NLDATEYQL}w. \]

**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.

```sas
options locale=en_US;
data _null_;  
dt = datetime();  
dy = date();  
put '+--- NLDATEYQL min=4 default=16 max=200 ----+
   16     T2 2019
   4     ****
   14     T2 2019
   32     2e trimestre 2019
   200
   2e trimestre 2019';  
put dy nldateyql.;  
put dy nldateyql6.;  
put dy nldateyql7.;  
put dy nldateyql18.;  
put dy nldateyql18.;  
run;
```
**NLDATEYQMw. Format**

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year's quarter value (Q1–Q4) using abbreviations.

**Syntax**

\[ \text{NLDATEYQMw.} \]

**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.

```sas
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;
```
**NLDATEYQS. Format**

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year and the year’s quarter value (1–4) with numbers and delimiters.

**Categories:**
CAS Date and Time

**Alignment:**
Left

**Syntax**

NLDATEYQS<sub>w</sub>.

**Syntax Description**

<sub>w</sub>

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.

```sas
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;
```
NLDATEYRw. Format

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year.

**Categories:** CAS
- Date and Time

**Alignment:** Left

### Syntax

**NLDATEYRw.**

### 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.

```sas
options locale=fr_FR;
data _null_;
  dt = datetime();
  dy = date();
  put '----- NLDATEYR min=4 default=6 max=200 -----
  2019/2
  19/2
  2019/2
  2019/2
run;
```

----- NLDATEYR min=4 default=6 max=200 -----
2019
19
2019
2019
**NLDATEYWw. Format**

Converts the SAS date value to the date value of the specified locale, and then writes the date value as the year and the week.

**Categories:** CAS  
Date and Time

**Alignment:** Left

**Syntax**

NLDATEYWw.

**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.

```sas
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     Week 14 2019
5      *****
8      W14 19
32     Week 14 2019
200    Week 14 2019
```
NLDATM*w. Format

Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime.

**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.

```plaintext
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. Format” on page 247
- “NLDATMTMw. Format” on page 256
- “NLDATMWw. Format” on page 258

---

**NLDATMAPw. Format**

Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as a datetime with a.m. or p.m.

**Categories:**
- CAS
- Date and Time

**Alignment:**
- Left

---

**Syntax**

NLDATMAPw.

**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 NLDATMAPw. format is similar to DATEAMPMw. except that the NLDATMAPw. 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.

```plaintext
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. Format” on page 246
• “NLDATMTMw. Format” on page 256
• “NLDATMWw. Format” on page 258

NLDATMDTw. Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month, day of the month and year.

Categories: CAS
Date and Time

Alignment: Left

Syntax

NLDATMDTw.

Syntax Description

w
specifies the width of the output field

Default 18
Range 10-200

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.

```sas
options locale=en_US;
data _null_;  
x=86400;  
put x nldatmdt.;  
run;
```
See Also

Formats:
- “NLDATMMDw. Format” on page 251

NLDATMLw. Format

Converts a SAS datetime value to the datetime string of the specified locale in the long representation of the date.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**

NLDATMLw.

**Syntax Description**

w

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
NLDATMM\(w\) Format

Converts a SAS datetime value to the datetime string of the specified locale in the medium representation of the date.

**Categories:** CAS

**Date and Time**

**Alignment:** Left

### Syntax

\[
\text{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 03:13:27 PM.

### Example

```plaintext
data _null_;  
dt = datetime();  
dy = date();  
put '++++ NLDATM\$ min=9 default=30 max=200 +++';  
put dt NLDATM\$.;  
put dt NLDATM\$19.;
```
NLDATMMDw. Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month and the day of the month.

Categories: CAS
Date and Time

Alignment: Left

Syntax

NLDATMMDw.

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. Format” on page 261
**NLDATMMDLw. Format**

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the full-length of the month and day of the month.

**Categories:**
- CAS
  - Date and Time

**Alignment:**
- Left

### Syntax

```
NLDATMMDLw.
```

### 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 nldatmmdl15.;  
put dt nldatmmdl19.;  
put dt nldatmmdl12.;  
put dt nldatmmdl200.;  
run;
```

```
+--- NLDATMMDL min=5 default=12 max=200 ----+
April 12
04/12
Apr 12
April 12
April 12
April 12
```
NLDATMMDMw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using abbreviations.

Categories: CAS
Date and Time

Alignment: Left

Syntax

NLDATMMDMw.

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.

```sas
data _null_;
  dt = datetime();
  dy = date();
  put '+---- NLDATMMDM min=5 default=9 max=200 ----+';
  put dt nldatmmmdm.;
  put dt nldatmmmdm5.;
  put dt nldatmmmdm9.;
  put dt nldatmmmdm200.;
run;
```

+---- NLDATMMDM min=5 default=9 max=200 ----+
Apr 12
04/12
Apr 12
Apr 12
NLDATMMDSw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and day of the month using numbers and delimiters.

**Categories:**
- CAS
  - Date and Time

**Alignment:** Left

**Syntax**

NLDATMMDSw.

**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.

```sas
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=5 max=200 ----+
04/12
04/12
04/12
04/12
```
NLDATMMNw. Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the name of the month.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**

NLDATMMNw.

**Syntax Description**

w specifies the width of the output field.

- **Default:** 9
- **Range:** 4–200

**Example**

This example uses the en_US locale option.

```sas
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

Converts a SAS datetime value to the datetime string of the specified locale in the short representation of the date.

**Categories:** CAS
Date and Time
**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
```

**NLDATMTM\(w\). Format**

Converts the time portion of a SAS datetime value to the time-of-day value of the specified locale, and then writes the value as a time of day.

**Categories:** CAS  
Date and 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.

```sql
options locale=en_US;
data one;
etvent=1361709583;
   put event nldatmtm.;
run;
```

12:39:43

This example specifies the German_Germany locale.

```sql
options locale=en_US;
data one;
etvent=1361709583;
   put event nldatmtm.;
run;
```

12.39 Uhr

**See Also**

**Formats:**

- “NLDATMw. Format” on page 246
- “NLDATMAPw. Format” on page 247
- “NLDATMWw. Format” on page 258
NLDATMTZw. Format
Converts the time portion of the SAS datetime value to the time of day and time zone of the specified locale.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**
NLDATMTZw.

**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.

```sas
options locale=fr_FR;
data test;
x=datetime();
put x=nldatmtz.;
run;
```

x=10 h 08 -0400

NLDATMWWw. Format
Converts SAS datetime values to the locale sensitive datetime string as the day of the week and the datetime.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**
NLDATMWWw.
**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.

**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.

```sas
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=\text{Monday, February 24, 2003 12:39:43 PM}
\[y=\text{Mon, Feb 24, 2003 12:39:43 PM}
\[z=\text{Monday, February 24, 2003 12:39:43 PM}

**See Also**

Formats:

- “NLDATMw. Format” on page 246
- “NLDATMAPw. Format” on page 247
- “NLDATMTMw. Format” on page 256
NLDATMWNw. Format

Converts a SAS datetime value to the datetime value of the specified locale, and then writes the value as the day of the week.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**
NLDATMWNw.

**Syntax Description**

w

specifies the width of the output field.

Default 9

Range 4–200

**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

Converts SAS date values of the specified locale to a day-of-week, datetime, and time zone value.

**Categories:** CAS
Date and Time
**Syntax**

NLDATMWZ_{w}.

**Syntax Description**

\( \textit{w} \)

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.

```plaintext
options locale=fr_FR;
data test;
x=datetime();
put x=nldatmwz.;
run;
```

```
x=mardi 16 avril 2019 10:35:06 -0400
```

---

**NLDATMYMw. Format**

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the month and year.

**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. Format” on page 251

NLDATMYMLw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.

Categories: CAS

Date and Time

Alignment: Left

Syntax

NLDATMYMLw.

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.

```sas
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=14 max=200 ---+ |
| April 2019 |
| 04/2019 |
| Apr 2019 |
| April 2019 |
| April 2019 |

NLDATMYMMw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the month and the year.

Categories: CAS
Date and Time

Alignment: Left

Syntax

NLDATMYMMw.

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.

```sas
data _null_; 
  dt = datetime();
  dy = date();
  put '+--- NLDATMYMM min=5 default=11 max=200 ----+';
  put dt nldatymmm put dt nldatymmm7.;
  put dt nldatymmm11.;
  put dt nldatymmm200.;
run;
```

+--- NLDATMYMM min=5 default=11 max=200 ----+
Apr 2019
04/2019
Apr 2019
Apr 2019

NLDATMYMSw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the month and year with numbers and a delimiter.

**Categories:** CAS  
Date and Time

**Alignment:** Left

**Syntax**

NLDATMYMSw.

**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.

```sas
data _null_; 
  dt = datetime();
```
dy = date();
put '++++ NLDATMYMS min=5 default=7 max=200 ----';
put dt nldatymys.;
put dt nldatymys5.;
put dt nldatymys7.;
put dt nldatymys200.;
run;

++++ NLDATMYMS min=5 default=7 max=200 ----
04/2019
04/19
04/2019
04/2019

NLDATMYQw. Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the quarter and the year.

Categories: CAS
Date and Time

Alignment: Left

Syntax

NLDATMYQw.

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.

```sas
options locale=en_US;
data _null_;
dy=today();
dt=datetime();
```
NLDATMYQLw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year’s quarter value (1–4) and the year.

| Category: | Date and Time |
| Alignment: | Left |

Syntax

NLDATMYQLw.

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.

```sas
options locale=en_US;
data _null_;  
  dt = datetime();
  dy = date();
```
NLDATMYQMw. Format

Converts a SAS date value to the date string of the specified locale, and then writes the date value as the year’s quarter (1–4) and then the year.

Categories: CAS
Date and Time

Alignment: Left

Syntax

NLDATMYQMw.

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 '++++ NLDATMYQL min=4 default=18 max=200 ++++'; 
put dt nldatmyq1.; 
put dt nldatmyq16.; 
put dt nldatmyq17.; 
put dt nldatmyq18.;  
put dt nldatmyq200.; 
run;  
```

++++ NLDATMYQL min=4 default=18 max=200 ++++
2nd quarter 2019
2019/2
Q2 2019
2nd quarter 2019
2nd quarter 2019
The following example uses the date April 16, 2019.

```sas
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;
```
NLDATMYRw. Format

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the year.

Categories: CAS  
Date and Time  

Alignment: Left  

Syntax

NLDATMYRw.

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

Converts the SAS datetime value to the datetime value of the specified locale, and then writes the value as the week number and the year.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**

NLDATMYWw.

**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.

```sas
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 ----+

<table>
<thead>
<tr>
<th>w</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>Week 16 2019</td>
</tr>
<tr>
<td>5</td>
<td>****</td>
</tr>
<tr>
<td>8</td>
<td>W16 19</td>
</tr>
<tr>
<td>32</td>
<td>Week 16 2019</td>
</tr>
<tr>
<td>200</td>
<td>Week 16 2019</td>
</tr>
</tbody>
</table>

NLDATMZWw. Format

Converts SAS datetime values to the locale-sensitive datetime string as datetime and time zone.

**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.

```plaintext
options locale=fr_FR;
data test;
x=datetime();
put x=nldatmz.;
run;
```

\(x=16\text{ avril 2019 11 h 40 -0400}\)

---

**NLMNIAED\(w.d\) 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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[
\begin{align*}
x &= \text{put}(-1234.56789, \text{nlmniaed32.2}); \\
y &= \text{put}(-1234.56789, \text{dollar32.2}); \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(AED1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57$</td>
</tr>
</tbody>
</table>

See Also

Format:

- “NLMNLAEDx.d Format” on page 308

---

NLMNIAUD\( w.d \) 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.
Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmniaud32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(AUD1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

See Also

Format:

- “NLMNLAUDw.d Format” on page 309

NLMNIBGNw.d Format

Writes the monetary format of the international expression for Bulgaria.

Categories: CAS Numeric

Alignment: Left

Syntax

NLMNIBGNw.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
Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[
x = \text{put}(-1234.56789, \text{nlmniibgn}32.2);
\]
\[
y = \text{put}(-1234.56789, \text{dollar}32.2);
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put (x=);</td>
<td>(BGN1,234.57)</td>
</tr>
<tr>
<td>put (y=);</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

See Also

Format:

- “\texttt{NLMNIBGNw.d Format}” on page 310

\texttt{NLMNIBRLw.d Format}

Writes the monetary format of the international expression for Brazil.

\begin{itemize}
  \item \textbf{Categories:} CAS
  \item \textbf{Numeric}
  \item \textbf{Alignment:} Left
\end{itemize}

Syntax

\texttt{NLMNIBRLw.d}

\textbf{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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnbrl32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(BRL1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

See Also

Format:
- “NLMNLBRLw.d Format” on page 311

NLMNICADw.d Format

Writes the monetary format of the international expression for Canada.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

Syntax

`NLMNICADw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.
x=put(-1234.56789,nlmnicad32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(CAD1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57$</td>
</tr>
</tbody>
</table>

**See Also**

Format:
- “NLMNLCADw.d Format” on page 312

---

**NLMNICHFw.d Format**

Writes the monetary format of the international expression for Liechtenstein and Switzerland.

**Categories:** CAS  
Numeric  

**Alignment:** Left

**Syntax**

NLMNICHFw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnicf32.2);
y=put(-1234.56789,dollar32.2);
See Also

Format:

- “NLMNLCHFw.d Format” on page 313

**NLMNICNYw.d Format**

Writes the monetary format of the international expression for China.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

**Syntax**

`NLMNICNYw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnicny32.2);
y=put(-1234.56789,dollar32.2);
```
### Statements

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(CNY1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

### See Also

**Format:**

- “NLMNLCNYw.d Format” on page 314

---

**NLMNICZKw.d Format**

Writes the monetary format of the international expression for the Czech Republic.

**Categories:** CAS, Numeric

**Alignment:** Left

### Syntax

NLMNICZKw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmniczk32.2);
y=put(-1234.56789,dollar32.2);
```
NLMNIDKKw.d Format

Writes the monetary format of the international expression for Denmark, Faroe Island, and Greenland.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNIDKKw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[ x=\text{put}(-1234.56789,\text{nlnidkk}32.2); \]
\[ y=\text{put}(-1234.56789,\text{dollar}32.2); \]
<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(DKK1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

See Also

Format:

- “NLMNLDKKw.d Format” on page 316

NLMNIEEKw.d Format

Writes the monetary format of the international expression for Estonia.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNIEEKw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\( x = \text{put}(-1234.56789, \text{nlmnieek32}.2); \)
\( y = \text{put}(-1234.56789, \text{dollar32}.2); \)
See Also

Format:

- “NLMNLEEKw.d Format” on page 317

NLMNIEGPw.d Format

W rites the monetary format of the international expression for Egypt.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNIEGPw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmniegp32.2);
y=put(-1234.56789,dollar32.2);
## 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

```plaintext
NLMNIEURw.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

In the following example, the `LOCASE= system option is set to `Locale=German_Germany`.

```plaintext
x=put(-1234.56789,nlmnieur32.2);
y=put(-1234.56789,nlmnieur32.2);
```

---

### Statements

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(E GP1, 234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57$</td>
</tr>
</tbody>
</table>

### See Also

**Format:**
- “NLMNLEGPw.d Format” on page 318
Statements | Results
---|---
put x=; | x=-1.234,57 EUR
put y=; | y=-1.234,57 €

See Also

Format:
- “NLMNLEURw.d Format” on page 319

**NLMNGBPw.d Format**

Writes the monetary format of the international expression for the United Kingdom.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

**Syntax**

\texttt{NLMNGBPw.d}

**Syntax Description**

\texttt{w}

specifies the width of the output field.

- Default: 12
- Range: 8–32

\texttt{d}

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

- Default: 2
- Range: 0–28

**Example**

In the following example, the LOCALE= system option is set to English_UnitedStates.

\begin{verbatim}
x=put(-1234.56789, nlmnigbp32.2);
y=put(-1234.56789, dollar32.2);
\end{verbatim}
See Also

Format:

- “NLMNLGBPw.d Format” on page 320

NLMNIHKDw.d Format

Writes the monetary format of the international expression for Hong Kong.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNIHKDw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnihkd32.2);
y=put(-1234.56789,dollar32.2);
NLMNIHRKw.d Format

Writes the monetary format of the international expression for Croatia.

**Categories:**
- CAS
- Numeric

**Alignment:** Left

**Syntax**

`NLMNIHRKw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmihrk32.2);
y=put(-1234.56789,dollar32.2);
```
### NLMNIHUFw.d Format

Writes the monetary format of the international expression for Hungary.

**Categories:** CAS, Numeric

**Alignment:** Left

### Syntax

\[ \text{NLMNIHUF} \_w.d \]

### 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

In the following example, the \text{LOCALE}\= system option is set to English\_UnitedStates.

\[
\begin{align*}
  x &= \text{put}\{\text{-1234.56789, nlmnihuf32.2}\}; \\
  y &= \text{put}\{\text{-1234.56789, dollar32.2}\};
\end{align*}
\]
See Also

Format:

- “NLMNLHUFw.d Format” on page 323
<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>IDR1,234.57</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

**See Also**

Format:
- “NLMNLIDRw.d Format” on page 324

---

**NLMNIILSw.d Format**

Writes the monetary format of the international expression for Israel.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

**Syntax**

NLMNIILSw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmniils32.2);
y=put(-1234.56789,dollar32.2);
See Also

Format:

- “NLMNLISw.d Format” on page 325

### NLMNIINRw.d Format

 Writes the monetary format of the international expression for India.

**Categories:** CAS

Numeric

**Alignment:** Left

#### Syntax

**NLMNIINRw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmniinr32.2);
y=put(-1234.56789,dollar32.2);
```
### Statements | Results
---|---
put x=; | (INR1,234.57)
put y=; | $-1,234.57

See Also

**Format:**
- “NLMNLINRw.d Format” on page 326

---

**NLMNIJPYw.d Format**

Writes the monetary format of the international expression for Japan.

**Category:** Numeric

**Alignment:** Left

**Syntax**

NLMNIJPY<var>W.D</var>

**Syntax Description**

<var>W</var>

specifies the width of the output field.

- **Default:** 12
- **Range:** 8–32

<var>D</var>

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

- **Default:** 0
- **Range:** 0–28

**Example**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnijpy32.2);
y=put(-1234.56789,dollar32.2);
```

---

**Statements** | **Results**
---|---
---|---
Statements | Results
---|---
put x=; | (JPY1,234.57)
put y=; | $-1,234.57

See Also

Format:
- “NLMNLJPYw.d Format” on page 327

**NLMNIKRWw.d Format**

Writes the monetary format of the international expression for South Korea.

**Categories:** CAS

| Alignment: | Left |

**Syntax**

NLMNIKRW<var>W.d</var>

**Syntax Description**

<var>W</var>

specifies the width of the output field.

| Default | 12 |
| Range | 8–32 |

<var>D</var>

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

| Default | 0 |
| Range | 0–28 |

**Example**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnikrw32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Statements | Results
---|---
put x=; | (KRW1,234.57)
put y=; | $-1,234.57

See Also

Format:

- “NLMNLKRw.d Format” on page 328

**NLMNILTLw.d Format**

Writes the monetary format of the international expression for Lithuania.

| Categories: | CAS  
| Numeric  |
| Alignment: | Left  |

**Syntax**

\[ \text{NLMNILTLw.d} \]

**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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmniltl32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
</table>
| -----+----- | 1-----+
### NLMNILVLw.d Format

Writes the monetary format of the international expression for Latvia.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

#### Syntax

NLMNILVLw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnilvl32.2);  
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(LTL1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

---

**See Also**

Format:
- “NLMNLLTLw.d Format” on page 329
Statements | Results
---|---
put x=; | (LVL1,234.57)
put y=; | $-1,234.57

See Also

Format:

- “NLMNLLVL.w.d Format” on page 330

NLMNIMOPw.d Format

Writes the monetary format of the international expression for Macau.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

**Syntax**

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

**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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[ x=\text{put}(-1234.56789,\text{nlnimop32.2}); \]
\[ y=\text{put}(-1234.56789,\text{dollar32.2}); \]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=put(-1234.56789,nlnimop32.2);</td>
<td>(LVL1,234.57)</td>
</tr>
<tr>
<td>y=put(-1234.56789,dollar32.2);</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
**Statements** | **Results**
--- | ---
`put x=;` | `(MOP1,234.57)`
`put y=;` | `$-1,234.57$`

### See Also

**Format:**
- “NLMNLMOPw.d Format” on page 331

---

### NLMNIMXNw.d Format

Writes the monetary format of the international expression for Mexico.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

### Syntax

`NLMNIMXNw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnimxn32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td><code>------+------</code></td>
<td><code>-------</code></td>
</tr>
</tbody>
</table>
Statements | Results
---|---
put x=; | (MXN1,234.57)
put y=; | $-1,234.57

See Also

Format:

- “NLMNLMXNw.d Format” on page 332

**NLMNIMYRw.d Format**

Writes the monetary format of the international expression for Malaysia.

**Categories:** CAS

Numeric

**Alignment:** Left

**Syntax**

NLMNIMYRw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

\( x=\text{put}(-1234.56789, \text{nmlminyr32.2}); \)

\( y=\text{put}(-1234.56789, \text{dollar32.2}); \)

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Statements | Results
---|---
put x=; | (MYR1,234.57)
put y=; | $-1,234.57

See Also

Format:
- “NLMNLMYRw.d Format” on page 333

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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\(x=\)put(-1234.56789,nlmninok32.2);
\(y=\)put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$-1,234.57$</td>
</tr>
</tbody>
</table>
### NLMNINZDw.d Format

Writes the monetary format of the international expression for New Zealand.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

**Syntax**

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

**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**

In the following example, the LOCATE= system option is set to English_UnitedStates.

\[
\begin{align*}
\text{x=put(-1234.56789,nlmninzd32.2);}
\text{y=put(-1234.56789,dollar32.2);} \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{put x=;}</td>
<td>{NOK1.234.57}</td>
</tr>
<tr>
<td>\text{put y=;}</td>
<td>$-1,234.57$</td>
</tr>
</tbody>
</table>

---

**See Also**

- “NLMNINOKw.d Format” on page 334
Statements | Results
---|---
put x=; | (NZD1,234.57)
put y=; | $-1,234.57

See Also

Format:

- “NLMNZNZDw.d Format” on page 335

NLMNIPLNWd Format

Writes the monetary format of the international expression for Poland.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Numeric</td>
</tr>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
</tbody>
</table>

Syntax

NLMNIPLNW.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnipln32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Statements</td>
<td>Results</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
</tr>
<tr>
<td>put x=;</td>
<td>(RUB1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

See Also

Format:

- “NLMNLPLNw.d Format” on page 336

NLMNIRUBw.d Format

Writes the monetary format of the international expression for Russia.

**Categories:**

- CAS
- Numeric

**Alignment:**

Left

**Syntax**

NLMNIRUBw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnirub32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(PLN1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNLRUBw.d Format” on page 337

NLMNISEKw.d Format

Writes the monetary format of the international expression for Sweden.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNISEKw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnisek32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(SEK1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNLSEKw.d Format” on page 338

NLMNISGDw.d Format

W rites the monetary format of the international expression for Singapore.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>CAS</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment:</td>
<td>Left</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

\[ \text{NLMNISGD}_{w.d} \]

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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[ x=\text{put}(-1234.56789,\text{nlmnisgd32.2}); \]
\[ y=\text{put}(-1234.56789,\text{dollar32.2}); \]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(SGD1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNLGDw.d Format” on page 339

**NLMNITHBw.d Format**

Writes the monetary format of the international expression for Thailand.

**Categories:** CAS

**Numeric**

**Alignment:** Left

**Syntax**

NLMNITHBw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnithb32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(THB1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNLTHBw.d Format” on page 340

NLMNITRYw.d Format
Writes the monetary format of the international expression for Turkey.

**Categories:** CAS
- Numeric

**Alignment:** Left

**Syntax**

NLMNITRYw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnitry32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(TRY1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNLTRYw.d Format” on page 341

NLMNITWDw.d Format

Writes the monetary format of the international expression for Taiwan.

Categories: CAS

Numeric

Alignment: Left

Syntax

NLMNITWDw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\( x = \)put(-1234.56789,nlmnitwd32.2);
\( y = \)put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(TWD1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNLTWDw.d Format” on page 342

NLMNIUSDw.d Format

Writers the monetary format of the international expression for Puerto Rico and the United States.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>CAS</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment:</td>
<td>Left</td>
<td></td>
</tr>
</tbody>
</table>

Syntax

NLMNIUSDw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmniusd32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(USD1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNUSDw.d Format” on page 343

NLMNIZARw.d Format

Writes the monetary format of the international expression for South Africa.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNIZARw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\( x=\text{put}(-1234.56789,\text{nlnizar}32.2); \)
\( y=\text{put}(-1234.56789,\text{dollar}32.2); \)

<table>
<thead>
<tr>
<th>Statements</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(ZAR1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNLZARw.d Format” on page 344

NLMNLAE dx.d Format

Writes the monetary format of the local expression for the United Arab Emirates.

Categories: CAS
           Numeric

Alignment: Left

Syntax

NLMNLAE dx.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlae32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(AED1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNIAEDw.d Format” on page 271

NLMNLAUDw.d Format

Writes the monetary format of the local expression for Australia.

Categories: CAS

Numeric

Alignment: Left

Syntax

NLMNLAUDw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[ x=\text{put}(-1234.56789, \text{nlmnlaud}32.2); \]

\[ y=\text{put}(-1234.56789, \text{dollar}32.2); \]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(AU$1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNIAUDw.d Format” on page 272

**NLMNLBGNw.d Format**

Writes the monetary format of the local expression for Bulgaria.

**Categories:**
- CAS
- Numeric

**Alignment:** Left

**Syntax**

NLMNLBGNw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnlbgn32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(BGN1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- "NLMNIBGNw.d Format" on page 273

**NLMNLBR\textit{w.d} Format**

Writes the monetary format of the local expression for Brazil.

**Categories:** CAS
- Numeric

**Alignment:** Left

**Syntax**

\texttt{NLMNLBR\textit{w.d}}

**Syntax Description**

\textit{w}

specifies the width of the output field.

- Default: 12
- Range: 8–32

\textit{d}

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

- Default: 2
- Range: 0–28

**Example**

In the following example, the \texttt{LOCALE=} system option is set to \texttt{English\_UnitedStates}.

\begin{verbatim}
x=put(-1234.56789,nlmnlbr132.2);
y=put(-1234.56789,dollar32.2);
\end{verbatim}

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{put x=;}</td>
<td>(R$1,234.57)</td>
</tr>
<tr>
<td>\texttt{put y=;}</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNIBRLw.d Format” on page 274

NLMNLCADw.d Format

Writes the monetary format of the local expression for Canada.

**Categories:** CAS

**Numeric**

**Alignment:** Left

**Syntax**

\texttt{NLMNLCADw.d}

**Syntax Description**

\texttt{w}

specifies the width of the output field.

- **Default:** 12
- **Range:** 8–32

\texttt{d}

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

- **Default:** 2
- **Range:** 0–28

**Example**

In the following example, the \texttt{LOCALE=} system option is set to \texttt{English_UnitedStates}.

\begin{verbatim}
x=put(-1234.56789,nlmnlcad32.2);
y=put(-1234.56789,dollar32.2);
\end{verbatim}

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put \texttt{x=};</td>
<td>\texttt{(CA$1,234.57)}</td>
</tr>
<tr>
<td>put \texttt{y=};</td>
<td>\texttt{$-1,234.57}$</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNICADw.d Format” on page 275

NLMNLCHFw.d Format

Writes the monetary format of the local expression for Liechtenstein and Switzerland.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNLCHFw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlchf32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>SFr.1,234.57</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

• “NLMNICHFw.d Format” on page 276

NLMNLCNYw.d Format

Writers the monetary format of the local expression for China.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNLCNYw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlcny32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(RMB1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNICNYw.d Format” on page 277

### NLMNLCZKw.d Format

Writes the monetary format of the local expression for the Czech Republic.

**Categories:** CAS

**Numeric**

**Alignment:** Left

### Syntax

`NLMNLCZKw.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

In the following example, the `LOCALE=` system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnlczk32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(CZK1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNICZKw.d Format” on page 278

NLMNLDKKw.d Format

Writes the monetary format of the local expression for Denmark, Faroe Island, and Greenland.

Categories: CAS

Numeric

Alignment: Left

Syntax

NLMNLDKKw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlk32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(kr1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNIDKKw.d Format” on page 279

NLMNLEEKw.d Format
Writes the monetary format of the local expression for Estonia.

Categories:
- CAS
- Numeric

Alignment:
- Left

Syntax

NLMNLEEKw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[
\begin{align*}
x &= \text{put}(-1234.56789, \text{nlmnleek}32.2); \\
y &= \text{put}(-1234.56789, \text{dollar}32.2);
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(Kr1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

---
See Also

Format:
- “NLMNIEEKw.d Format” on page 280

NLMNLEGpw.d Format
Writes the monetary format of the local expression for Egypt.

Categories:
- CAS
- Numeric

Alignment: Left

Syntax
NLMNLEGpw.d

Syntax Description

\( \text{w} \)
- specifies the width of the output field.
  - Default: 12
  - Range: 8–32

\( \text{d} \)
- specifies the number of digits to the right of the decimal point in the numeric value.
  - Default: 3
  - Range: 0–28

Example
In the following example, the LOCALE= system option is set to English_UnitedStates.

\[
\begin{align*}
\text{x} &= \text{put}(-1234.56789, \text{nlmnlegp}_32.2) ; \\
\text{y} &= \text{put}(-1234.56789, \text{dollar}_32.2) ; \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(EGP1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>


See Also

Format:

• “NLMNIEGPw.d Format” on page 281

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

NLMNLEURw.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

In the following example, the LOCALE= system option is set to German_Germany.

x=put(-1234.56789,nlmnieur32.2);
y=put(-1234.56789,nlmnleur32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>x=-1.234,57 EUR</td>
</tr>
<tr>
<td>put y=;</td>
<td>y=-1.234,57 €</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNIEURw.d Format” on page 282

NLMNLGBPw.d Format

Writers the monetary format of the local expression for the United Kingdom.

Categories: CAS

Numeric

Alignment: Left

Syntax

NLMNLGBPw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlgbp32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(£1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNIGBPw.d Format” on page 283

NLMNLHKDw.d Format
Writes the monetary format of the local expression for Hong Kong.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNLHKDw.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
In the following example, the LOCALE= system option is set to English_UnitedStates.

\( x=\text{put(-1234.56789,nlmnlhkhd32.2)}; \)
\( y=\text{put(-1234.56789,dollar32.2)}; \)

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(HK$1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNIHKDw.d Format” on page 284

NLMNLHRKw.d Format
W rites the monetary format of the local expression for Croatia.

Categories: CAS
          Numeric

Alignment: Left

Syntax

NLMNLHRKw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[ x=\text{put}(-1234.56789,\text{nlmnlnrk32.2}); \]
\[ y=\text{put}(-1234.56789,\text{dollar32.2}); \]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{put } x=;</td>
<td>(Kn1,234.57)</td>
</tr>
<tr>
<td>\text{put } y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNIHRKw.d Format” on page 285

NLMNLHUFw.d Format

Writs the monetary format of the local expression for Hungary.

**Categories:** CAS

**Numeric**

**Alignment:** Left

Syntax

NLMNLHUFw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[
\begin{align*}
x &= \text{put}(-1234.56789, \text{nlmnlhuf32.2}); \\
y &= \text{put}(-1234.56789, \text{dollar32.2});
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(Ft1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNHUFw.d Format” on page 286

NLMNLIDRw.d Format

Writes the monetary format of the local expression for Indonesia.

Categories:
- CAS
- Numeric

Alignment:
- Left

Syntax

NLMNLIDRw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnlidr32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(Rp1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNIIDRw.d Format” on page 287

### NLMNLILSw.d Format

Writes the monetary format of the local expression for Israel.

**Categories:** CAS, Numeric

**Alignment:** Left

#### Syntax

`NLMNLILSw.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

In the following example, the `LOCALE=` system option is set to `English_UnitedStates`.

```plaintext
x=put(-1234.56789,nlmnlils32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(ILS1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNIILSw.d Format” on page 288

NLMNLINRw.d Format

Writes the monetary format of the local expression for India.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNLINRw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlirn32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=put(-1234.56789,nlmnlirn32.2);</td>
<td>(INR1,234.57)</td>
</tr>
<tr>
<td>y=put(-1234.56789,dollar32.2);</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNIINRw.d Format” on page 289

NLMNLJPYw.d Format
Writes the monetary format of the international expression for Japan.

Categories: CAS
Numeric
Alignment: Left

Syntax

NLMNLJPYw.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
In the following example, the LOCALE= system option is set to English_UnitedStates.

\[
\begin{align*}
x & = \text{put}(-1234.56789, \text{nlmnljpy}32.2) ; \\
y & = \text{put}(-1234.56789, \text{dollar}32.2) ;
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(JPY1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNIIJPYw.d Format” on page 290

NLMNLKRWw.d Format
Wrtes the monetary format of the local expression for South Korea.

Categories:
- CAS
- Numeric

Alignment: Left

Syntax

NLMNLKRWw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlkrw32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(KRW1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
### See Also

Format:
- “NLMNIKRw.d Format” on page 291

---

### NLMNLLTLw.d Format

Writes the monetary format of the local expression for Lithuania.

**Categories:** CAS, Numeric

**Alignment:** Left

---

### Syntax

\[ \text{NLMNLLTLw.d} \]

**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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\( x=\text{put}(-1234.56789, \text{nlmnlltl32.2}); \)
\( y=\text{put}(-1234.56789, \text{dollar32.2}); \)

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(LT1, 234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

---
See Also

Format:
- “NLMNITLw.d Format” on page 292

NLMNLLVLw.d Format
Writs the monetary format of the local expression for Latvia.

Categories: CAS
          Numeric

Alignment: Left

Syntax

NLMNLLVLw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnllvl132.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(LS1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNILVLw.d Format” on page 293

NLMNLMOPw.d Format

Writes the monetary format of the local expression for Macau.

**Categories:** CAS, Numeric

**Alignment:** Left

Syntax

NLMNLMOPw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnlmop32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(P1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
• “NLMNIMOPw.d Format” on page 294

NLMNLMXNw.d Format
Writesthe monetary format of the local expression for Mexico.

Categories: CAS
          Numeric

Alignment: Left

Syntax

NLMNLMXNw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlmxn32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(MX$1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNIMXNw.d Format” on page 295

### NLMNLMYRw.d Format

Writes the monetary format of the local expression for Malaysia.

#### Categories:
- CAS
- Numeric

#### Alignment:
- Left

### Syntax

`NLMNLMYRw.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

In the following example, the `LOCALE=` system option is set to English_UnitedStates.

```
x=put(-1234.56789,nlmnlmyr32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>$-1,234.57$</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57$</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNIMYRw.d Format” on page 296

NLMNLNOKw.d Format

Writes the monetary format of the local expression for Norway.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alignment</th>
<th>Left</th>
</tr>
</thead>
</table>

Syntax

NLMNLNOKw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[
\begin{align*}
x &= \text{put}(-1234.56789, \text{nlnlnok}32.2); \\
y &= \text{put}(-1234.56789, \text{dollar}32.2);
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(kr1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNINOKw.d Format” on page 297

NLMNLNZDw.d Format

Writes the monetary format of the local expression for New Zealand.

**Categories:** CAS, Numeric  
**Alignment:** Left

**Syntax**

NLMNLNZDw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnlnzd32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(NZ$1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMBINZDw.d Format” on page 298

NLMNLPLNw.d Format

Writes the monetary format of the local expression for Poland.

**Categories:** CAS

**Alignment:** Left

**Syntax**

NLMNLPLNw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

\[ x=\text{put}(-1234.56789,\text{nlmnlpln}32.2); \]
\[ y=\text{put}(-1234.56789,\text{dollar}32.2) \]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(PLN1, 234.57</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1, 234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNINPLNw.d Format” on page 299

NLMNLRUBw.d Format

Writes the monetary format of the local expression for Russia.

Categories:  CAS
          Numeric

Alignment:  Left

Syntax

NLMNLRUBw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlnlrub32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(RUB1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNIRUBw.d Format” on page 300

---

**NLMNLSEKw.d Format**

Writes the monetary format of the local expression for Sweden.

**Categories:** CAS

Numeric

**Alignment:** Left

---

**Syntax**

NLMNLSEKw.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**

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnlsek32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(kr1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$\text{-}1,234.57$</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNISEKw.d Format” on page 301

NLMNLSGDw.d Format

Writes the monetary format of the local expression for Singapore.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNLSGDw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

\nx=put(-1234.56789,nlmnlsgd32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(SG$1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNISGDw.d Format” on page 302

NLMNLTHBw.d Format

Writes the monetary format of the local expression for Thailand.

Categories: CAS
          Numeric

Alignment: Left

Syntax

NLMNLTHBw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlthb32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(THB1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNITHBw.d Format” on page 303

NLMNLTRYw.d Format

Writes the monetary format of the local expression for Turkey.

- **Categories:** CAS, Numeric
- **Alignment:** Left

Syntax

NLMNLTRYw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

```plaintext
x=put(-1234.56789,nlmnltry32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(YTL1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNITRYw.d Format” on page 304

NLMNLTW Dw.d Format
Writes the monetary format of the local expression for Taiwan.

Categories: CAS
Numeric
Alignment: Left

Syntax
NLMNLTW Dw.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
In the following example, the LOCALE= system option is set to English_UnitedStates.

\( x = \)put(-1234.56789,nlmnltdw32.2);
\( y = \)put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put ( x = ) ;</td>
<td>(NT$1,234.57)</td>
</tr>
<tr>
<td>put ( y = ) ;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:
- “NLMNITWDw.d Format” on page 305

NLMNLUSDw.d Format

Writers the monetary format of the local expression for Puerto Rico and the United States.

Categories: CAS
Numeric

Alignment: Left

Syntax

NLMNLUSDw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmnlusd32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(US$1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>-$1,234.57</td>
</tr>
</tbody>
</table>
See Also

Format:

- “NLMNIUSDw.d Format” on page 306

NLMNLZARw.d Format
Writes the monetary format of the local expression for South Africa.

Categories: CAS
Numeric
Alignment: Left

Syntax

NLMNLZARw.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

In the following example, the LOCALE= system option is set to English_UnitedStates.

x=put(-1234.56789,nlmlzar32.2);
y=put(-1234.56789,dollar32.2);

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(R1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>
NLMNYw.d Format

Writers the monetary format of the local expression in the specified locale using local currency.

**Categories:** CAS

**Alignment:** Left

**Syntax**

NLMNYw.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 NLMNYw.d informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLMNYw.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 NLMNYw.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 NLMNYw.d and NLMNYIw.d formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNYIw.d format uses three-letter international currency codes, such as USD, while NLMNYw.d format uses local currency symbols, such as $.

The NLMNYw.d format is similar to the DOLLARw.d format except that the NLMNYw.d format is locale-specific.
Example

In the following example, the LOCALE= system option is set to English_UnitedStates.

```sas
x=put(-1234.56789,nlmny32.2);
y=put(-1234.56789,dollar32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>($1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

See Also

Format:
- “NLMNYIw.d Format” on page 346

Informats:

NLMNYIw.d Format

Writes the monetary format of the international expression in the specified locale.

**Categories:**
- CAS
- Numeric

**Alignment:** Left

**Syntax**

```sas
NLMNYIw.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 NLMNYIw.d informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The NLMNYIw.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 NLMNYIw.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 NLMNYw.d and NLMNYIw.d formats write the monetary format with locale-dependent thousands and decimal separators. However, the NLMNYIw.d format uses three-letter international currency codes, such as USD, while NLMNYw.d format uses local currency symbols, such as $.

Example
In the following example, the LOCALE= system option is set to English_UnitedStates.

\[
\begin{align*}
x &= \text{put}(-1234.56789, \text{nlmnyi32.2}); \\
y &= \text{put}(-1234.56789, \text{nlmny32.2}); \\
z &= \text{put}(-1234.56789, \text{dollar32.2});
\end{align*}
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>(USD1,234.57)</td>
</tr>
<tr>
<td>put y=;</td>
<td>($1,234.57)</td>
</tr>
<tr>
<td>put z=;</td>
<td>$-1,234.57</td>
</tr>
</tbody>
</table>

See Also

Format:
- “NLNUMw.d Format” on page 345

Informats:

NLNUMw.d Format
Writes the numeric format of the local expression in the specified locale.
Syntax

`NLNUMw.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 `NLNUMw.d` informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The `NLNUMw.d` format writes numeric values by using the thousands separator and the decimal separator that is used by the locale.

**Comparisons**

The `NLNUMw.d` format writes the numeric value with locale-dependent thousand and decimal separators. The `NLNUMIw.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 `NLNUMw.d` format uses an algorithm that prints the thousands-separator characters whenever possible, even if some decimal precision is lost.

**Example**

```plaintext
x=put(-1234356.7891,nlnum32.2);
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>options LOCALE=English_UnitedStates;</td>
<td>-1,234,356.79</td>
</tr>
<tr>
<td>put x=;</td>
<td></td>
</tr>
</tbody>
</table>
Statements | Results
---|---
options LOCALE=German_Germany; | -1.234,356,79
put x=;

See Also

Format:

- “NLNUMIw.d Format” on page 349

Informat:


### NLNUMIw.d Format

Writes the numeric format of the international expression in the specified locale.

**Categories:**
- CAS
  - Numeric

**Alignment:**
- Left

**Syntax**

`NLNUMIw.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 `NLNUMIw.d` informat reads integer binary (fixed-point) values, including negative values that are represented in two's-complement notation. The `NLNUMIw.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

\[x=\text{put}\left(-1234356.7891,\text{nlnumi32.2}\right);\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>options LOCALE=English_UnitedStates;</td>
<td>-1,234,356.79</td>
</tr>
<tr>
<td>put (x=;)</td>
<td></td>
</tr>
<tr>
<td>options LOCALE=German_Germany;</td>
<td>-1,234,356.79</td>
</tr>
<tr>
<td>put (x=;)</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Format:
- “NLNUMw.d Format” on page 347

Informats:

### NLPCT\(w.d\) Format

Writes percentage data of the local expression in the specified locale.

**Categories:** CAS
- Numeric

**Alignment:** Left

**Syntax**

\[\text{NLPCT}w.d\]
**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**

\[
x = \text{put(-12.3456789, nlpct32.2)};
y = \text{put(-12.3456789, nlpcti32.2)};
z = \text{put(-12.3456789, percent32.2)};
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-----------------</td>
</tr>
<tr>
<td>options LOCALE=English_UnitedStates; put x=; put y=; put z=;</td>
<td>-1,234.57%</td>
</tr>
<tr>
<td>options LOCALE=German_Germany; put x=; put y=; put z=;</td>
<td>-1,234.57%</td>
</tr>
</tbody>
</table>

**See Also**

**Format:**

- “NLPCTI\( w.d \) Format” on page 352
NLPCTIw.d Format

Writes percentage data of the international expression in the specified locale.

**Categories:**
- CAS
- Numeric

**Alignment:** Left

**Syntax**

\texttt{NLPCTIw.d}

**Syntax Description**

\texttt{w}

specifies the width of the output field.

- Default: 6
- Range: 4–32

\texttt{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 NLPCTIw.d format writes percentage data of the international expression in the specified locale. The NLPCTw.d format writes the percentage value with locale-dependent thousand and decimal separators. The NLPCTIw.d format writes the percentage value with a comma (,) as thousands separator and a period (.) as a decimal separator.

The NLPCTw.d format is similar to the PERCENTw.d format except the NLPCTIw.d format is locale-specific.

**Example**

In the following example, the \texttt{LOCALE=} system option is set to English_UnitedStates.

\begin{verbatim}
x=put(-12.3456789,nlpcti32.2);
y=put(-12.3456789,percent32.2);
\end{verbatim}
Statements | Results
--- | ---
-----+--------
put x=; | -1,234.57%
put y=; | [ 1234.57]

See Also

Format:
- “NLPCTw.d Format” on page 350

Informats:

### NLPCTNw.d Format

Produces percentages, using a minus sign for negative values.

**Categories:** CAS

**Numeric:**

**Alignment:** Right

#### Syntax

NLPCTNw.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**

\[
x = -0.02;
\]

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{put x nlpctn6;} )</td>
<td>( x = -2% )</td>
</tr>
<tr>
<td>( \text{put x percentn6;} )</td>
<td>( x = -2% )</td>
</tr>
</tbody>
</table>

**NLPCTP\( w.d \) Format**

Writes locale-specific numeric values as percentages.

**Categories:** CAS
- Numeric

**Alignment:** Right

**Syntax**

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

**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.
  - **Default:** 0
  - **Range:** 0–31
  - **Requirement:** must be less than \( w \)
Details

The NLPCTP\texttt{w.d} format multiplies values by 100, formats them, and adds a percent sign (%) to the end of the formatted value. The NLPCTP\texttt{w.d} format is similar to the The \texttt{PERCENTw.d} format except that the thousands separator and decimal symbol for the NLPCTP\texttt{w.d} format is locale-specific.

Example

\begin{verbatim}
x=-0.02;
put x nlpctp6.;  
put x percent6.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x nlpctp6.;</td>
<td>-2%</td>
</tr>
<tr>
<td>put x percent6.;</td>
<td>( 2%)</td>
</tr>
</tbody>
</table>

\textbf{NLPVALUEw.d Format}

Writes p-values of the local expression in the specified locale.

\begin{tabular}{l}
\textbf{Categories:} CAS
\textbf{Numeric} \\
\textbf{Alignment:} Left
\end{tabular}

\textbf{Syntax}

\texttt{NLPVALUEw.d}

\textbf{Syntax Description}

\texttt{w}

specifies the width of the output field.

\begin{itemize}
\item Default 6
\item Range 3–32
\end{itemize}

\texttt{d}

specifies to divide the number by 10\textsuperscript{d}. If the data contains decimal separators, the \texttt{d} value is ignored.

\begin{itemize}
\item Default 4
\item Range 1–30
\end{itemize}

\textbf{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 pvalue1.1 +5 x nlpvalue1.1;  
    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 Format” on page 378

NLSTRMONw.d Format

Writes the month name in the specified locale.

Categories:  CAS

Numeric

Alignment:  Left

Syntax

NLSTRMONw.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.
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.

<table>
<thead>
<tr>
<th>Statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data <em>null</em>;</td>
</tr>
<tr>
<td>monnum = 1 ; /* January=1, December=12 */</td>
</tr>
<tr>
<td>put monnum NLSTRMON20. ;</td>
</tr>
<tr>
<td>put monnum NLSTRMON20.1; /* decimal .1 specified use abbreviation. */</td>
</tr>
<tr>
<td>put monnum NLSTRMON20.2;</td>
</tr>
<tr>
<td>put monnum NLSTRMON20.3;</td>
</tr>
<tr>
<td>run;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>January</td>
</tr>
<tr>
<td>Jan</td>
</tr>
<tr>
<td>JANUARY</td>
</tr>
<tr>
<td>JAN</td>
</tr>
</tbody>
</table>

NLSTRQTRw.d Format

Writes a numeric value as the quarter-of-the-year in the specified locale.

- **Categories:** CAS Numeric
- **Alignment:** Left
Syntax

NLSTRQTRw.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.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data <em>null</em> ;</td>
<td>1st quarter</td>
</tr>
<tr>
<td>qtrnum = 1 ; /* January=1, December=12 */ put qtrnum NLSTRQTR20. ;</td>
<td>Q1</td>
</tr>
<tr>
<td>put qtrnum NLSTRQTR20.1 ; /* decimal .1 specified use abbreviation. */</td>
<td>1ST QUARTER</td>
</tr>
<tr>
<td>put qtrnum NLSTRQTR20.2;</td>
<td>Q1</td>
</tr>
<tr>
<td>put qtrnum NLSTRQTR20.3; run;</td>
<td></td>
</tr>
</tbody>
</table>


**NLSTRWKw.d Format**

Writes a numeric value as the day-of-the-week in the specified locale.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

**Syntax**

NLSTRWK\texttt{w.d}

**Syntax Description**

\texttt{w}

- specifies the width of the output field

  - Default: 20
  - Range: 1–200

\texttt{d}

- specifies the following:
  - 00000001: write abbreviated form.
  - 00000010: write capitalized form.

  - Default: 0
  - Range: 0–3

**Details**

The NLSTRWK\texttt{w.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.
NLTIMAPw Format

Converts a SAS time value to the time value of a specified locale, and then writes the value as a time value with a.m. or p.m. NLTIMAP also converts SAS date-time values.

**Categories:** CAS
Date and Time

**Alignment:** Left

**Syntax**

NLTIMAPw.

**Syntax Description**

w

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4–200</td>
</tr>
</tbody>
</table>

**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 NLTIMAPw format is similar to the TIMEAMPMw format except that the NLTIMAPw format is locale-specific.
Example

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>options locale=English_UnitedStates;</td>
<td>4:24:43 PM</td>
</tr>
<tr>
<td>put time nltimap.;</td>
<td></td>
</tr>
<tr>
<td>options locale=German_Germany;</td>
<td>4:24:43 nachm</td>
</tr>
<tr>
<td>put time nltimap14.;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Format:

- “NLTIMEw. Format” on page 361

NLTIMEw. Format

Converts a SAS time value to the time value of the specified locale, and then writes the value as a time value. NLTIME also converts SAS date-time values.

**Categories:**
CAS
Date and Time

**Alignment:**
Left

**Syntax**

NLTIMEw.

**Syntax Description**

w

specifies the width of the input field.

Default 20

Range 10–200

**Comparisons**

The NLTIMEw. format is similar to the TIMEw. format except that the NLTIMEw. format is locale-specific.
Example

These examples use the input value of 59083, which is the SAS date-time value that corresponds to 4:24:43 p.m.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>options locale=English_UnitedStates;</td>
<td>4:24:43</td>
</tr>
<tr>
<td>put time nctime.;</td>
<td></td>
</tr>
<tr>
<td>options locale=German_Germany;</td>
<td>16.24</td>
</tr>
<tr>
<td>put time nctime.;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Format:

- “NLTIMAPw. Format” on page 360

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= System Option” in SAS System Options: Reference.

**Syntax**

NUMXw.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.
Details
The NUMXw.d format writes numeric values with a comma in place of the decimal point.

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

Example
```
data one;
  input x;
data lines;
  896.48
  64.89
  3064.10
;run;

data two;
  set one;
  put x numx10.2;
run;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>896.48</td>
<td>896,48</td>
</tr>
<tr>
<td>64.89</td>
<td>64,89</td>
</tr>
<tr>
<td>3064.10</td>
<td>3064,10</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “w.d Format” on page 411

Informat:
- “NUMXw.d Informat” on page 569
OCTALw. Format

Converts numeric values to octal representation.

**Categories:**
- CAS
- Numeric

**Alignment:**
- Left

**Syntax**

`OCTALw.`

**Syntax Description**

`w`

- Specifies the width of the output field.

  **Default:** 3
  **Range:** 1–24

**Details**

If necessary, the OCTALw. format converts numeric values to integers before displaying them in octal representation.

**Comparisons**

OCTALw. converts numeric values to octal representation. The SOCTALw. format converts character values to octal representation.

**Example**

```plaintext
data one;
  x=3592;
  put x octal6.;
run;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3592</td>
<td>007010</td>
</tr>
</tbody>
</table>

**ODDSRw.d Format**

Writes odds ratios.

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

Syntax

ODDSR\(w.d\)

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 \(10^{-d-2}\)–\(\text{eps}\) 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

```plaintext
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;

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---+----1</td>
<td></td>
</tr>
<tr>
<td>0.00001 &lt;0.001</td>
<td></td>
</tr>
<tr>
<td>0.0001 &lt;0.001</td>
<td></td>
</tr>
<tr>
<td>0.001 0.001</td>
<td></td>
</tr>
<tr>
<td>0.01   0.010</td>
<td></td>
</tr>
<tr>
<td>0.1    0.100</td>
<td></td>
</tr>
<tr>
<td>1      1.000</td>
<td></td>
</tr>
<tr>
<td>10     10.000</td>
<td></td>
</tr>
<tr>
<td>100    100.000</td>
<td></td>
</tr>
<tr>
<td>1000   &gt;999.999</td>
<td></td>
</tr>
<tr>
<td>I      &gt;999.999</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>-l     *********</td>
<td></td>
</tr>
</tbody>
</table>

**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= System Option” in SAS System Options: Reference.

See: “PD Format: UNIX” in SAS Companion for UNIX Environments
Syntax

PD<sub>w</sub>.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<sup>d</sup>. This argument is optional.
  - Default: 0
  - Range: 0–31

**Details**

Different operating environments store packed decimal values in different ways. However, the PD<sub>w</sub>.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<sub>w</sub>.d format writes missing numerical data as –0. When the PD<sub>w</sub>.d informat reads a –0, it stores it as 0.

**Comparisons**

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

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

**Example**

```plaintext
data a;
  x=123;
  y=put(x,pd4.);
  put y $hex8.;
```
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

W r i t e s p a c k e d J u l i a n d a t e v a l u e s i n t h e hexadec i m a l f o r m a t \textit{yyyydddF} for IBM.

\begin{itemize}
\item \textbf{Category:} Date and Time
\item \textbf{Restriction:} This format is not supported in a DATA step that runs in CAS.
\end{itemize}

\subsection*{Syntax}

\texttt{PDJULGw.}

\subsection*{Syntax Description}

\begin{itemize}
\item \texttt{w} specifies the width of the output field.
\item Default: 4
\item Range: 3–16
\end{itemize}

\subsection*{Details}

The PDJULGw. format writes SAS date values in the form \textit{yyyydddF}.

\begin{itemize}
\item \textit{yyyy} is the 2-byte representation of the four-digit Gregorian year.
\item \textit{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).
\item \texttt{F} is the half-byte that contains all binary 1s, which assigns the value as positive.
\end{itemize}

\textbf{Note:} SAS interprets a two-digit year as part of the 100-year span that is defined by the \texttt{YEARCUTOFF=} system option.

\subsection*{Example}

\begin{verbatim}
date = '15sep2018'd;
juldate = put(date, pdjulg4.);
put juldate $hex8.;
run;
\end{verbatim}
SAS writes the following results to the log:

```
20182587F
```

See Also

Formats:
- “JULDAYw. Format” on page 202
- “JULIANw. Format” on page 204
- “PDJULIw. Format” on page 369

Functions:
- “DATEJUL Function” in SAS Functions and CALL Routines: Reference
- “JULDATE Function” in SAS Functions and CALL Routines: Reference

Informats:
- “JULIANw. Informat” on page 562
- “PDJULGw. Informat” on page 573
- “PDJULIw. Informat” on page 575

System Options:
- “YEARCUTOFF= System Option” in SAS System Options: Reference

---

**PDJULIw. Format**

Writes packed Julian date values in the hexadecimal format `ccyydddF` for IBM.

**Category:** Date and Time

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

---

**Syntax**

```
PDJULIw.
```

**Syntax Description**

`w`

specifies the width of the output field.

**Default:** 4

**Range:** 3–16

---

**Details**

The PDJULIw. format writes SAS date values in the form `ccyydddF`. 
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 PDJULIw. 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

```sas
data _null_;
  date = '15jun2018'd;
  juldate = put(date,pdjuli4.);
  put juldate $hex8.;
run;
```

SAS writes the following results to the log:

```
0118258F
```

Example 2

```sas
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. Format” on page 202
- “JULIANw. Format” on page 204
- “PDJULGw. Format” on page 368

Functions:
- “DATEJUL Function” in SAS Functions and CALL Routines: Reference
PERCENTw.d Format

**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= System Option” in *SAS System Options: Reference*.

**Syntax**

PERCENTw.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 PERCENTw.d format multiplies values by 100, formats them the same as the BESTw.d format, and adds a percent sign (%) to the end of the formatted value. Negative values are enclosed in parentheses.

Example
``` SAS
data one;
  input gain;
datalines;
  .1
  1.2
  -0.05
;
run;

data two;
  set one;
  put @10 gain percent10.;
run;
```

<table>
<thead>
<tr>
<th>Value of gain</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>10%</td>
</tr>
<tr>
<td>1.2</td>
<td>120%</td>
</tr>
<tr>
<td>-0.05</td>
<td>( 5%)</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “PERCENTNw.d Format” on page 372

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 STDIEEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV= System Option” 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**

```plaintext
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

<table>
<thead>
<tr>
<th>Value of gain</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.1</td>
<td>-10%</td>
</tr>
<tr>
<td>.2</td>
<td>20%</td>
</tr>
<tr>
<td>.8</td>
<td>80%</td>
</tr>
<tr>
<td>-0.05</td>
<td>-5%</td>
</tr>
<tr>
<td>-6.3</td>
<td>-630%</td>
</tr>
</tbody>
</table>

### See Also

**Format:**
- “PERCENTw.d Format” on page 371

### 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 Format: Windows” in SAS Companion for Windows

### Syntax

PIB<sup>w.d</sup>

#### Syntax Description

**w**

specifies the width of the output field.

- **Default:** 1  
- **Range:** 1–8

**d**

specifies to multiply the number by 10<sup>d</sup>. This argument is optional.

- **Default:** 0  
- **Range:** 0–10
Details

All values are treated as positive. PIBw.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 33.

Comparisons

• Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIBw.d format treats all values as positive and includes the sign bit as part of the value.

• The PIBw.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 PIBw.d format is the same as the IBw.d format except that PIBw.d treats all values as positive values.

• The IBw.d and PIBw.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 IBRw.d and PIBRw.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 33.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

Example

```plaintext
data _null_;  
x=12;  
y=put(x,pib1.);  
put y $hex2.;  
run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0C</td>
</tr>
</tbody>
</table>

* 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 Format” on page 376

---

### 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

```markdown
PIBRw.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. PIBRw.d writes positive integer binary values that have been generated by and for Intel and DEC operating environments. Use PIBRw.d to write positive integer binary data from Intel or DEC environments in other operating environments. The PIBRw.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 33.

---

### Comparisons

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIBRw.d format treats all values as positive and includes the sign bit as part of the value.
- The PIBRw.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 PIBw.d and PIBRw.d formats are equivalent.
- The IBw.d and PIBw.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 IBRw.d and PIBRw.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 33.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

**Example**

```plaintext
data _null_;  
x=128;  
y=put(x,pibr2.);  
   put y $hex4.;  
run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>8000</td>
</tr>
</tbody>
</table>

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

**See Also**

Informats:
- “PIBw.d Informat” on page 578

---

**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 STDIEEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV= System Option” in SAS System Options: Reference.
Syntax

\texttt{PKw.d}

**Syntax Description**

\textit{w} specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1–16</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0–10</td>
</tr>
</tbody>
</table>

**Details**

Each byte of unsigned packed decimal data contains two digits.

**Comparisons**

The \texttt{PKw.d} format is similar to the \texttt{PDw.d} format except that \texttt{PKw.d} does not write the sign in the low-order byte.

**Example**

```plaintext
data _null_;  
x=128;  
y=put(x,pk4.);  
put y $hex8.;  
run;
```

<table>
<thead>
<tr>
<th>Value of \textit{y}</th>
<th>Result $^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0000000128</td>
</tr>
</tbody>
</table>

$^*$ 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.
Syntax

\texttt{PVALUEw.d}

\textbf{Syntax Description}

\textit{w}

specifies the width of the output field.

\begin{itemize}
  \item Default: 6
  \item Range: 3–32
\end{itemize}

\textit{d}

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

\begin{itemize}
  \item Default: the minimum of 4 and \texttt{w–2}
  \item Range: 1–30
  \item Requirement: \textit{d} must be less than \textit{w}.
\end{itemize}

\textbf{Comparisons}

The \texttt{PVALUEw.d} format follows the rules for the \textit{w.d} format, except in these conditions:

\begin{itemize}
  \item if the value \( x \) is such that \( 0 <= x < 10^{-d} \), \( x \) is printed as “\(<.0...01\)” with \( d-1 \) zeros
  \item missing values are printed as “.” unless you specify a different character by using the \texttt{MISSING=} system option
\end{itemize}

\textbf{Example}

\begin{verbatim}
data one;
  input x;
  datalines;
  .05
  .000001
  0
  .0123456
; run;

data two;
  set one;
  put x pvalue6.4;
\end{verbatim}
QTRw. Format

Writes date values as the quarter of the year.

Categories: CAS
           Date and Time

Alignment: Right

Syntax

QTRw.

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

  put x qtr.;  1
See Also

Formats:
• “QTRRw. Format” on page 381

QTRRw. Format

W rites date values as the quarter of the year in Roman numerals.

Categories: CAS
Date and Time
Alignment: Right

Syntax
QTRRw.

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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x qtrr.;</td>
<td>III</td>
</tr>
</tbody>
</table>

See Also

Formats:
• “QTRw. Format” on page 380
**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 Format: Windows” in SAS Companion for Windows
- “RBw.d Format: z/OS” in SAS Companion for z/OS

**Syntax**

`RBw.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 `RBw.d` format writes numeric data in the same way that SAS stores the data. Because it requires no data conversion, `RBw.d` is the most efficient method for writing data with SAS.

**Note:** Different operating environments store real binary values in different ways. However, `RBw.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:
<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>RB4.</td>
<td>RB8.</td>
</tr>
<tr>
<td>Fortran</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>double</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-1</td>
<td>COMP-2</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>E</td>
<td>D</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
data one;
  x=128;
  y=put(x,rb8.);
  put y $hex16.;
run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>000000000006040</td>
</tr>
</tbody>
</table>

* 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**

`ROMANw.`

**Syntax Description**

`w`

specifies the width of the output field.

- **Default:** 6
- **Range:** 2–32
Details

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

Example

```plaintext
data one;
  input x;
datalines;
2018
  52.3
;
run;
data two;
  set one;
  put x roman10.;
run;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018</td>
<td>MMXVIII</td>
</tr>
<tr>
<td>52.3</td>
<td>LII</td>
</tr>
</tbody>
</table>

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

S370FFw.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 S370FFw.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, S370FFw.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, S370FFw.d behaves like the w.d format.
- On all other systems, S370FFw.d performs the same role for numeric data that the $EBCDICw. format does for character data.

Example

```plaintext
data _null_;  
x=12345;    
y=put(x,s370ff5.);   
put y $hex10.;  
run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>F1F2F3F4F5</td>
</tr>
</tbody>
</table>

* The result is the hexadecimal representation for the integer.

See Also

Formats:
- "$EBCDICw. Format" on page 98
- “w.d Format” on page 411

S370FIBw.d Format

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

<table>
<thead>
<tr>
<th>Category:</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
</tbody>
</table>

Syntax

S370FIBw.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 33.

**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 33.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

**Example**

```plaintext
data _null_
  x=128;
  y=put(x,s370fib4.);
  put y $hex8.;
run;
```

<table>
<thead>
<tr>
<th>Value of ( y )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>- - - - - - - - -</td>
<td>1</td>
</tr>
<tr>
<td>Value of y</td>
<td>Result</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>128</td>
<td>00000080</td>
</tr>
</tbody>
</table>

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

See Also

Formats:
- “S370FIBUw.d Format” on page 387
- “S370FPIBw.d Format” on page 391

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

S370FIBUw.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 S370FIBUw.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. S370FIBUw.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 S370FIBUw.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 33.

Comparisons

• The S370FIBUw.d format is equivalent to the COBOL notation PIC 9(n) BINARY, where n is the number of digits.

• The S370FIBUw.d format is the same as the S370FIBw.d format except that the S370FIBUw.d format always uses the absolute value instead of the signed value.

• The S370FPIBw.d format writes all negative numbers as FFs, whereas the S370FIBUw.d format writes the absolute value.

• S370FPIBw.d, S370FIBUw.d, and S370FIBw.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 33.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

Example

```plaintext
data one;
  input x;
  datalines;
  245
  -245
;
run;

data two;
  set one;
  y=put(x,s370fibu2.);
  put y $hex4.;
run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>F5</td>
</tr>
<tr>
<td>-245</td>
<td>F5</td>
</tr>
</tbody>
</table>

* 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 Format” on page 385
- “S370FPIBw.d Format” on page 391

S370FDPw.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= System Option” in SAS System Options: Reference.

Syntax

S370FDPw.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 S370FDPw.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 and Packed Decimal Notation

<table>
<thead>
<tr>
<th>Language</th>
<th>Packed Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FPD4.</td>
</tr>
<tr>
<td>PL/I</td>
<td>FIXED DEC(7,0)</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-3 PIC S9(7)</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>PL4</td>
</tr>
</tbody>
</table>

### Example

```r
data _null_;
x=128;
y=put(x,s370fpd4.);
put y $hex8.;
runcn;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0000128C</td>
</tr>
</tbody>
</table>

* 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= System Option” in SAS System Options: Reference.

### Syntax

**S370FPDUw.d**

### Syntax Description

`w`

specifies the width of the output field.
\( d \)
specifies to multiply the number by \(10^d\). This argument is optional.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1–16</td>
</tr>
</tbody>
</table>

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

```plaintext
data one;
  input x;
data lines;
  123  -123
;run;
data two;
set one;
y=put(x,s370fpdu4.);
put y $hex8.;run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>123F</td>
</tr>
<tr>
<td>-123</td>
<td>123F</td>
</tr>
</tbody>
</table>

* 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.

S370FPIB\(w,d\) Format

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

Category: Numeric
Syntax

S370FPIBw.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. S370FPIBw.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 S370FPIBw.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 33.

Comparisons

• If you use SAS on an IBM mainframe, S370FPIBw.d and PIBw.d are identical.

• The S370FPIBw.d format is the same as the S370FIBw.d format except that the S370FPIBw.d format treats all values as positive values.

• S370FPIBw.d, S370FIBUw.d, and S370FIBw.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 33.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.
Example

data _null_;  
x=12;  
y=put(x,s370fpib1.);  
put y $hex2.;  
run;

| Value of y | Result *
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>0C</td>
</tr>
</tbody>
</table>

* 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 Format” on page 385
- “S370FIBUw.d Format” on page 387

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**

S370FRBw.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<sub>w</sub>.<sub>d</sub> 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:

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
<tr>
<td>Fortran</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-1</td>
<td>COMP-2</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>double</td>
</tr>
</tbody>
</table>

Example

```plaintext
data _null_
  input x;
  y=put(x,s370frb6.);
  put y $hex8.;
  datalines;
  128
  -128
  ;
  run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>42800000</td>
</tr>
<tr>
<td>-123</td>
<td>C2800000</td>
</tr>
</tbody>
</table>

* 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.
**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:

<table>
<thead>
<tr>
<th>Language</th>
<th>Zoned Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FZD3.</td>
</tr>
<tr>
<td>PL/I</td>
<td>PICTURE '99T'</td>
</tr>
<tr>
<td>COBOL</td>
<td>PIC S9(3) DISPLAY</td>
</tr>
<tr>
<td>assembler</td>
<td>ZL3</td>
</tr>
</tbody>
</table>

**Example**

```
data one;
  input x;
```
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= System Option” in SAS System Options: Reference.

Syntax

S370FZDLw.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

* 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.
Details

Use S370FZDLw.d in other operating environments to write zoned decimal leading-sign data in the same format as on an IBM mainframe computer.

Comparisons

- The S370FZDLw.d format is similar to the S370FZDw.d format except that the S370FZDLw.d format displays the sign of the number in the first byte of the formatted output.
- The S370FZDLw.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 *
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>CLF2F3</td>
</tr>
<tr>
<td>-123</td>
<td>DLF2F3</td>
</tr>
</tbody>
</table>

* 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

S370FZDSw.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**

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

```r
data two;
  set one;
  y=put(x,s370fzds3.);
  put y $hex6.;
run;
```

<table>
<thead>
<tr>
<th>Value of (y)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>4EF1F2F3</td>
</tr>
<tr>
<td>-123</td>
<td>60F1F2F3</td>
</tr>
</tbody>
</table>

* 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= System Option” in SAS System Options: Reference.

**Syntax**

\[
S370FZDTw.d
\]

**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 S370FZDTw.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 S370FZDTw.d format is similar to the S370FZDSw.d format except that the S370FZDTw.d format displays the sign of the number at the end of the formatted output.
- The S370FZDTw.d format is equivalent to the COBOL notation PIC S9(\(n\)) DISPLAY SIGN TRAILING SEPARATE, where \(n\) is the number of digits.

**Example**

```sas
data one;
  input x;
  datalines;
  123
```
-123
;
run;
data two;
set one;
y=put(x,s370fzdt4.);
put y $hex8.;
run;

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>F1F2F34E</td>
</tr>
<tr>
<td>-123</td>
<td>F1F2F360</td>
</tr>
</tbody>
</table>

* 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= System Option” in SAS System Options: Reference.

**Syntax**

```
S370FZDUw.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 S370FZDUw.d in other operating environments to write unsigned zoned decimal data in the same format as on an IBM mainframe computer.

Comparisons
• The S370FZDUw.d format is similar to the S370FZDw.d format, except that the S370FZDUw.d format always uses the absolute value of the number.
• The S370FZDUw.d format is equivalent to the COBOL notation PIC 9(n) DISPLAY, where n is the number of digits.

Example
```plaintext
data one;
  input x;
datalines;
  123
  -123
;
run;
data two;
  set one;
  y=put(x,s370fzdu3.);
  put y $hex6.;
run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>F1F2F3</td>
</tr>
<tr>
<td>-123</td>
<td>F1F2F3</td>
</tr>
</tbody>
</table>

* 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.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction:</td>
<td>This format is not supported in a DATA step that runs in CAS.</td>
</tr>
</tbody>
</table>

Syntax

```
SSNw.
```

Syntax Description
w
   specifies the width of the output field.
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

```sas
data one;
  id=263878439;
  put id ssn.;
run;
```

<table>
<thead>
<tr>
<th>Value of id</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>263878439</td>
<td>263-87-8439</td>
</tr>
</tbody>
</table>

TIME\( .d \) Format

Writes time values as hours, minutes, and seconds in the form \( \text{hh:mm:ss.ss} \).

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= System Option” 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\( .d \)

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 \( \text{TIME}w.d \) format writes SAS time values in the form \( hh:mm:ss.ss \):

- \( hh \) is an integer.
  
  Note: If \( hh \) is a single digit, \( \text{TIME}w.d \) places a leading blank before the digit. For example, the \( \text{TIME}w.d \) 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 \( \text{TIME}w.d \) format is similar to the \( \text{HHMM}w.d \) format except that \( \text{TIME}w.d \) includes seconds.

The \( \text{TIME}w.d \) format writes a leading blank for a single-hour digit. The \( \text{TOD}w.d \) 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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mytime time.;</td>
<td>16:24:43</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mytime time.;</td>
<td>24:00:00</td>
</tr>
</tbody>
</table>

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;).

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put diff time8.;</td>
<td>-1:00:00</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “HHMMw.d Format” on page 195
- “HOURw.d Format” on page 197
- “MMSSw.d Format” on page 211
- “TODw.d Format” on page 406

Functions:
- “HOUR Function” in SAS Functions and CALL Routines: Reference
- “MINUTE Function” in SAS Functions and CALL Routines: Reference
- “SECOND Function” in SAS Functions and CALL Routines: Reference
- “TIME Function” in SAS Functions and CALL Routines: Reference

Informats:
TIMEAMP\text{\texttt{Mw.d}} Format

Writes time and datetime values as hours, minutes, and seconds in the form \textit{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= System Option” in SAS System Options: Reference.

Syntax

\texttt{TIMEAMPM\_w.d}

\textbf{Syntax Description}

\textit{w}

specifies the width of the output field.

Default \hspace{1em} 11

Range \hspace{1em} 2–20

\textit{d}

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

Default \hspace{1em} 0

Range \hspace{1em} 0–19

Requirement \hspace{1em} must be less than \textit{w}

\textbf{Details}

The TIMEAMP\text{\texttt{Mw.d}} format writes SAS time values and SAS datetime values in the form \textit{hh:mm:ss.ss} with AM or PM. The letters \textit{h}, \textit{m}, and \textit{s} are defined as follows:

\textit{hh}

is an integer that represents the hour.

\textit{mm}

is an integer that represents the minutes.

\textit{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

```sas
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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put begin timeampm3.;</td>
<td>PM</td>
</tr>
<tr>
<td>put begin timeampm5.;</td>
<td>4 PM</td>
</tr>
<tr>
<td>put begin timeampm8.;</td>
<td>4:24 PM</td>
</tr>
<tr>
<td>put begin timeampm11.;</td>
<td>4:24:43 PM</td>
</tr>
</tbody>
</table>

### See Also

**Formats:**

- “TIME\( w.d \) Format” on page 402

### TOD\( w.d \) 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 STDIEEEE, the output that is written using this format might differ slightly from previous releases. For more
information, see “DECIMALCONV= System Option” 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', it formats the value as 1:00:00.

Syntax

\texttt{TODw.d}

Syntax Description

\textbf{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.

\textbf{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 \textit{d must be less than \textit{w}}.

Details

The \texttt{TODw.d} format writes SAS time and datetime values in the form \textit{hh:mm:ss.ss}.

\textit{hh}

is an integer that represents the hour.

\textit{mm}

is an integer that represents the minutes.

\textit{ss.ss}

is the number of seconds to two decimal places.

Comparisons

The \texttt{TODw.d} format writes a leading zero for a single-hour digit. The \texttt{TIMEw.d} format and the \texttt{HHMMw.d} format write a leading blank for a single-hour digit.

Examples

\textbf{Example 1}

\begin{verbatim}
data one;
  mydt1='24aug2018 2:20:23 pm'dt;
\end{verbatim}
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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydt1 tod5. ;</td>
<td>14:20</td>
</tr>
<tr>
<td>put mydt2 tod9. ;</td>
<td>01:30:00</td>
</tr>
<tr>
<td>put mydt3 tod9. ;</td>
<td>08:54:43</td>
</tr>
</tbody>
</table>

**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;

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin = 32083 ;</td>
<td></td>
</tr>
<tr>
<td>put begin tod9. ;</td>
<td>08:54:43</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “HHMMw.d Format” on page 195
- “TIMEw.d Format” on page 402
- “TIMEAMPMw.d Format” on page 405

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

**Informats:**
- “TIMEw. Informat” on page 610
VAXRBw.d Format

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

Categories: CAS
Numeric

Alignment: Right

Syntax

VAXRBw.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 VAXRBw.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 VAXRBw.d and RBw.d formats are identical.

Example

```
data one;
  x=1;
  y=put(x,vaxrb8.);
  put y $hex16.;
run;
```

<table>
<thead>
<tr>
<th>Value of y</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80400000000000</td>
</tr>
</tbody>
</table>

* 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**

VMSNZ\textit{n}w.d

**Syntax Description**

\textit{w}

specifies the width of the output field.

\begin{center}
\begin{tabular}{l|l}
\hline
Default & 1 \\
Range & 1–32 \\
\hline
\end{tabular}
\end{center}

\textit{d}

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

**Details**

The VMSNZ\textit{n}w.d format is similar to the ZD\textit{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 VMSNZ\textit{n}w.d format.

\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
Desired Digit & Special Character & Desired Digit & Special Character \\
\hline
0 & 0 & -0 & p \\
\hline
1 & 1 & -1 & q \\
\hline
2 & 2 & -2 & r \\
\hline
3 & 3 & -3 & s \\
\hline
4 & 4 & -4 & t \\
\hline
5 & 5 & -5 & u \\
\hline
6 & 6 & -6 & v \\
\hline
7 & 7 & -7 & w \\
\hline
\end{tabular}
\end{center}
If the value to be formatted is too large to fit in a field of the specified width, the VMSZNW.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 vmszn5.1;
run;

SAS writes the following results to the log:

```
1234
12340
123400
01234
```

**See Also**

**Formats:**
- “ZDw.d Format” on page 451

**Informats:**
- “VMSZNW.d Informat” on page 615

**w.d Format**

Writes standard numeric data one digit per byte.

**Categories:** CAS

**Numeric**

**Alignment:** Right

**Alias:** Fw.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= System Option” in SAS System Options: Reference.

Syntax

\[ w.d \]

Syntax Description

\( w \)

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Range</th>
<th>1–32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tip</td>
<td>Allow enough space to write the value, the decimal point, and a minus sign, if necessary.</td>
</tr>
</tbody>
</table>

\( d \)

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

<table>
<thead>
<tr>
<th>Range</th>
<th>0–31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>must be less than ( w )</td>
</tr>
<tr>
<td>Tip</td>
<td>If ( d ) is 0 or you omit ( d ), ( w.d ) writes the value without a decimal point.</td>
</tr>
</tbody>
</table>

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 BESTw. 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 \( Zw.d \) format is similar to the \( w.d \) format except that \( Zw.d \) pads right-aligned output with 0s instead of blanks.

Example

```sas
data one;
x=23.45;
put x 6.3; run;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.45</td>
<td>23.450</td>
</tr>
</tbody>
</table>
**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 and Time
- **Alignment:** Right
- **Restriction:** Use the NLDATEw. format on the CAS server instead of the WEEKDATEw format.

**Syntax**

`WEEKDATEw.`

**Syntax Description**

`w`

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>3–37</td>
</tr>
</tbody>
</table>

**Details**

The WEEKDATEw. 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 WEEKDATEw. format is the same as the WEEKDATXw. format, except that WEEKDATXw. 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.

```plaintext
data one;
  mydate='14jun2018'd;
  put mydate weekdate3.;
  put mydate weekdate9.;
  put mydate weekdate15.;
  put mydate weekdate17.;
run;
```
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 and Time

**Alignment:** Right
Syntax

WEEKDATX\(w\).

Syntax Description

\(w\)

specifies the width of the output field.

Default 29

Range 3–37

Details

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

\(dd\)

is an integer that represents the day of the month.

\(yy\) or \(yyyy\)

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

If \(w\) is too small to write the complete day of the week and month, 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.

```sas
data one;
    mydate='14jun2018'd;
    put mydate weekdatx. ;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate weekdatx.;</td>
<td>Thursday, 14 June 2018</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DATEw. Format” on page 150
“WEEKDAYw. Format” on page 413
“YYMMDDw. Format” on page 430

**Functions:**

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

**Informats:**

- “DATEw. Informat” on page 537
- “DDMMYYw. Informat” on page 540
- “MMDDYYw. Informat” on page 565
- “YYMMDDw. Informat” on page 626

---

**WEEKDAYw. Format**

Writes date values as the day of the week.

**Categories:** CAS
Date and Time

**Alignment:** Right

**Syntax**

`WEEKDAYw.`

**Syntax Description**

`w`

specifies the width of the output field.

Default 1

Range 1–32

**Details**

The WEEKDAYw. 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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate weekday.;</td>
<td>7</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DOWNAMEw. Format” on page 163

WEEKUw. Format

Writes a week number in decimal format by using the U algorithm.

- **Category:** Date and Time
- **Alignment:** Left
- **Restriction:** This format is not supported in a DATA step that runs in CAS.

Syntax

WEEKUw.

**Syntax Description**

w

specifies the width of the output field.

- **Default:** 11
- **Range:** 2–200

Details

The WEEKUw. format writes a week-number format. The WEEKUw. 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:

<table>
<thead>
<tr>
<th>Width</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-4</td>
<td>www</td>
<td>w01</td>
</tr>
<tr>
<td>5-6</td>
<td>yywww</td>
<td>18w01</td>
</tr>
<tr>
<td>7-8</td>
<td>yywwwdd</td>
<td>18W0101</td>
</tr>
<tr>
<td>9-10</td>
<td>yyyywwwdd</td>
<td>2018w0101</td>
</tr>
<tr>
<td>11-200</td>
<td>yyyy-Www-dd</td>
<td>2018-W01-01</td>
</tr>
</tbody>
</table>

**Comparisons**

The WEEKWw. 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 WEEKWw. 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 WEEKUw. 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**

```sas
   data one;
      sasdate='16apr2018'd;
      put sasdate weeku3.;
      put sasdate weeku5.;
      put sasdate weeku7.;
      put sasdate weeku9.;
      put sasdate weeku11.;
   run;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate weeku3.;</td>
<td>W15</td>
</tr>
<tr>
<td>put sasdate weeku5.;</td>
<td>18W15</td>
</tr>
<tr>
<td>put sasdate weeku7.;</td>
<td>18W1502</td>
</tr>
<tr>
<td>put sasdate weeku9.;</td>
<td>2018W1502</td>
</tr>
<tr>
<td>put sasdate weeku11.;</td>
<td>2018-W15-02</td>
</tr>
</tbody>
</table>
See Also

Formats:
- “WEEKVw. Format” on page 419
- “WEEKWw. Format” on page 421

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

Informats:
- “WEEKUw. Informat” on page 618
- “WEEKVw. Informat” on page 620
- “WEEKWw. Informat” on page 622

WEEKVw. Format

Writes a week number in decimal format by using the V algorithm.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
<tr>
<td>Restriction:</td>
<td>This format is not supported in a DATA step that runs in CAS.</td>
</tr>
</tbody>
</table>

Syntax

WEEKVw.

Syntax Description

w

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>2–200</td>
</tr>
</tbody>
</table>

Details

The WEEKVw. 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:
Comparisons

The `WEEKVw.` 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 `WEEKWw.` 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 `WEEKUw.` 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

```sas
data one;
  sasdate='16apr2018'd;
  put sasdate weekv3.;
  put sasdate weekv5.;
  put sasdate weekv7.;
  put sasdate weekv9.;
  put sasdate weekv11.;
run;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate weekv3.;</td>
<td>W37</td>
</tr>
<tr>
<td>put sasdate weekv5.;</td>
<td>18W15</td>
</tr>
<tr>
<td>put sasdate weekv7.;</td>
<td>18W1502</td>
</tr>
<tr>
<td>put sasdate weekv9.;</td>
<td>2018W1502</td>
</tr>
<tr>
<td>put sasdate weekv11.;</td>
<td>2018-W15-02</td>
</tr>
</tbody>
</table>

See Also

Formats:
Functions:
- “WEEK Uw. Format” on page 417
- “WEEK Ww. Format” on page 421

Informats:
- “WEEK Uw. Informat” on page 618
- “WEEK Vw. Informat” on page 620
- “WEEK Ww. Informat” on page 622

### WEEK Ww. Format

Writes a week number in decimal format by using the W algorithm.

- **Category:** Date and Time
- **Alignment:** Left
- **Restriction:** This format is not supported in a DATA step that runs in CAS.

#### Syntax

```
WEEKWw.
```

#### Syntax Description

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

  - **Default:** 11
  - **Range:** 2–200

#### Details

The WEEK Ww. 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:

<table>
<thead>
<tr>
<th>Width</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>3–4</td>
<td>Www</td>
<td>w01</td>
</tr>
<tr>
<td>5–6</td>
<td>yyWww</td>
<td>18W01</td>
</tr>
<tr>
<td>7–8</td>
<td>yyWwwdd</td>
<td>18W0101</td>
</tr>
</tbody>
</table>
Comparisons

The WEEKVw. 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 WEEKWw. 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 WEEKUw. 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;  

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate weekv3.;</td>
<td>W16</td>
</tr>
<tr>
<td>put sasdate weekv5.;</td>
<td>18W16</td>
</tr>
<tr>
<td>put sasdate weekv7.;</td>
<td>18W1601</td>
</tr>
<tr>
<td>put sasdate weekv9.;</td>
<td>2018W1601</td>
</tr>
<tr>
<td>put sasdate weekv11.;</td>
<td>2018-W16-01</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WEEKUw. Format” on page 417
- “WEEKVw. Format” on page 419

Functions:
- “WEEK Function” in SAS Functions and CALL Routines: Reference
**WORDDATEw. Format**

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

**Syntax**

\texttt{WORDDATEw.}

**Syntax Description**

\texttt{w} specifies the width of the output field.

- **Default:** 18
- **Range:** 3–32

**Details**

The \texttt{WORDDATEw.} format writes SAS date values in the form *month-name dd, yyyy.*

- \texttt{dd} is an integer that represents the day of the month.
- \texttt{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 \texttt{WORDDATEw.} format is the same as the \texttt{WORDDATXw.} format, except that \texttt{WORDDATXw.} prints \texttt{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.

\begin{verbatim}
   data one;
      mydate='14jun2018'd;
      put mydate worddate3.;
      put mydate worddate9.;
\end{verbatim}
SAS Statement | Result
---|---
| 1 2

put mydate worddate12.;
put mydate worddate20.;
run;

See Also

Formats:
- "WORDDATXw. Format" on page 424

WORDDATXw. Format

WOrds date values as the day, the name of the month, and the year in the form \textit{dd month-name yyyy}.

\begin{itemize}
  \item \textbf{Category:} Date and Time
  \item \textbf{Alignment:} Right
  \item \textbf{Restriction:} This format is not supported in a DATA step that runs in CAS.
\end{itemize}

\subsection*{Syntax}

\texttt{WORDDATXw}.

\subsection*{Syntax Description}

\textit{w}

\textit{w} specifies the width of the output field.

\begin{itemize}
  \item \textbf{Default:} 18
  \item \textbf{Range:} 3–32
\end{itemize}

\subsection*{Details}

The \texttt{WORDDATXw} format writes SAS date values in the form \textit{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 WORDDATXw. format is the same as the WORDDATEw. format, except that WORDDATEw. 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.

```sas
data one;
  mydate='14jun2018'd;
  put mydate worddatx.;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate worddatx.;</td>
<td>14 June 2018</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “**WORDDATEw. Format**” on page 423

---

**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**

**WORDFw.**
**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**

```plaintext
data one;
  value=2.5;
  put value wordf15.;
run;
```

<table>
<thead>
<tr>
<th>Value of value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>two and 50/100</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “WORDS\( w \). Format” on page 426

**WORDS\( w \). 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 `WORDFw` format, except that `WORDS w` prints fractions as words instead of numbers.

Example

```plaintext
data one;
  value=2.5;
  put value words25.;
run;
```

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>two and fifty hundredths</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “`WORDFw. Format`” on page 425

YEARw. Format

Writes date values as the year.

Categories: CAS
Syntax

YEAR\(w\).

Syntax Description

\(w\)

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>2–32</td>
</tr>
<tr>
<td>Tip</td>
<td>If (w) is less than 4, the last two digits of the year are printed. Otherwise, the year value is printed as four digits.</td>
</tr>
</tbody>
</table>

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

```sas
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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate year2. ;</td>
<td>18</td>
</tr>
<tr>
<td>put mydate year4. ;</td>
<td>2018</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DTYEAR\(w\) Format” on page 168
YYMMw. Format

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.

**Categories:** CAS
  Date and Time

**Alignment:** Right

**Syntax**

YYMMw.

**Syntax Description**

w

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5–32</td>
</tr>
</tbody>
</table>

**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 YYMMw. format writes SAS date values in the form <yy>yyMmm.

<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**

```sas
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.; | 20182M0610

**See Also**

**Formats:**
- “MMYYw. Format” on page 212
- “YYMMxw. Format” on page 434

**YYMMDDw. Format**

Writes date values in the form `yyymmdd` or `<yy>yy-mm-dd`, where a hyphen is the separator and the year appears as either two or four digits.

**Categories:**
- CAS
- Date and Time

**Alignment:** Right

**Syntax**

`YYMMDDw`.

**Syntax Description**

`w`

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
<th>Interaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2–10</td>
<td>When <code>w</code> has a value from 2 to 5, the date appears with as much of the year and the month as possible. When <code>w</code> is 7, the date appears as a two-digit year without hyphens.</td>
</tr>
</tbody>
</table>
Details

The YYMMDw. format writes SAS date values in one of these forms:

*yyymmdd*

<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 YYMMDDr. format.

Example

```sas
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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate yymmdd2.;</td>
<td>18</td>
</tr>
<tr>
<td>put mydate yymmdd3.;</td>
<td>18</td>
</tr>
<tr>
<td>put mydate yymmdd4.;</td>
<td>1804</td>
</tr>
<tr>
<td>put mydate yymmdd5.;</td>
<td>18-04</td>
</tr>
<tr>
<td>put mydate yymmdd6.;</td>
<td>180403</td>
</tr>
<tr>
<td>put mydate yymmdd7.;</td>
<td>180403</td>
</tr>
<tr>
<td>SAS Statement</td>
<td>Result</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>put mydate yymmdd8.;</td>
<td>18-04-03</td>
</tr>
<tr>
<td>put mydate yymmdd10.;</td>
<td>2018-04-03</td>
</tr>
</tbody>
</table>

See Also

**Formats:**
- “DATEw. Format” on page 150
- “DDMMYYw. Format” on page 156
- “MMDDYYw. Format” on page 207
- “YYMMDDxw. Format” on page 432

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

**Informats:**
- “DATEw. Informat” on page 537
- “DDMMYYw. Informat” on page 540
- “MMDDYYw. Informat” on page 565

### 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 and Time

**Alignment:** Right

**Syntax**

`YYMMDDxw.`
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 YYMMDxw. format writes SAS date values in one of these forms:

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.

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;
The following examples use the input value of 21318, which is the SAS date value that corresponds to May 14, 2018.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate yymmddc5.;</td>
<td>18:05</td>
</tr>
<tr>
<td>put mydate yymmddd8.;</td>
<td>18-05-14</td>
</tr>
<tr>
<td>put mydate yymmddp10.;</td>
<td>2018.05.14</td>
</tr>
<tr>
<td>put mydate yymmddn8.;</td>
<td>20180514</td>
</tr>
</tbody>
</table>

---

See Also

Formats:
- “DATEw. Format” on page 150
- “DDMMYYxw. Format” on page 158
- “MMDDYYxw. Format” on page 209
- “YYMMDDw. Format” on page 430

Functions:
- “DAY Function” in SAS Functions and CALL Routines: Reference
- “MDY Function” in SAS Functions and CALL Routines: Reference
- “MONTH Function” in SAS Functions and CALL Routines: Reference
- “YEAR Function” in SAS Functions and CALL Routines: Reference

Informats:
- “YYMMDDw. Informat” on page 626

---

YYMMxw. Format

Writes date values in the form <yy>yy-mm 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
            Date and Time
Syntax

YYMM\textsubscript{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.

<table>
<thead>
<tr>
<th>Default</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>5–32</td>
</tr>
</tbody>
</table>

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\textsubscript{xw} format writes SAS date values in one of these forms:

\(<\text{yy}\textgreater yy\textless mm\textgreater >\)

\(<\text{yy}\textless yy\textgreater X\text{mm}\textgreater >\)

The letters are defined as follows:

- \(<\text{yy}\textgreater >\text{yy}\)
  
  is a two-digit or four-digit integer that represents the year.

- \(x\)
  
  is a specified separator.
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.

```sas
data one;
  mydate='14may2018'd;
  put mydate yymmc5.;
  put mydate yymmd.;
  put mydate yymmn4.;
  put mydate yymmp8.;
  put mydate yymms10.;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate yymmc5.;</td>
<td>18:05</td>
</tr>
<tr>
<td>put mydate yymmd.;</td>
<td>2018-05</td>
</tr>
<tr>
<td>put mydate yymnn4.;</td>
<td>1805</td>
</tr>
<tr>
<td>put mydate yymmp8.;</td>
<td>2018.05</td>
</tr>
<tr>
<td>put mydate yymms10.;</td>
<td>2018/05</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “MMYYxw. Format” on page 214
- “YYMMw. Format” on page 429

YYMOnw. Format

Writes date values in the form `yyymm` or `yyyyymmm`.

**Categories:** CAS

Date and Time

Numeric

**Alignment:** Right
Syntax
YYMONw.

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 YYMONw. 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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-----+----1</td>
</tr>
<tr>
<td>put mydate yymon6. ;</td>
<td>18JUN</td>
</tr>
<tr>
<td>put mydate yymon7. ;</td>
<td>2018JUN</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “MMYYw. Format” on page 212
YYQw. Format

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.

**Categories:** CAS
Date and Time

**Alignment:** Right

**Syntax**

YYQw.

**Syntax Description**

\( w \)

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Range</strong></td>
<td>4–32</td>
</tr>
</tbody>
</table>

**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 YYQw. format writes SAS date values in the form <yy>yyQq.

- \(<yy>yy\) 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.

```sas
data one;
  mydate='14jun2018'd;
  put mydate yyq.;
  put mydate yyq4.;
  put mydate yyq5.;
  put mydate yyq6.;
  put mydate yyq10.;
run;
```
<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate yyq.;</td>
<td>2018Q2</td>
</tr>
<tr>
<td>put mydate yyq4.;</td>
<td>18Q2</td>
</tr>
<tr>
<td>put mydate yyq5.;</td>
<td>18Q2</td>
</tr>
<tr>
<td>put mydate yyq6.;</td>
<td>2018Q2</td>
</tr>
<tr>
<td>put mydate yyq10.;</td>
<td>2018Q2</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “YYQxw. Format” on page 439
- “YYQRw. Format” on page 441

**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 and Time

**Alignment:**
- Right

**Syntax**

`YYQxw`

**Syntax Description**

The character `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 YYQxw. 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;
<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate yyqc4.;</td>
<td>18:3</td>
</tr>
<tr>
<td>put mydate yyqd.;</td>
<td>2018-3</td>
</tr>
<tr>
<td>put mydate yyqn3.;</td>
<td>183</td>
</tr>
<tr>
<td>put mydate yyqp6.;</td>
<td>2018.3</td>
</tr>
<tr>
<td>put mydate yyqs8.;</td>
<td>2018/3</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “YYQw. Format” on page 438
- “YYQRxw. Format” on page 442

**YYQRw. Format**

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.

**Categories:** CAS
Date and Time

**Alignment:** Right

**Syntax**

**YYQRw.**

**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 YYQRw. format writes SAS date values in the form `<yy>yyQqr`. 
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.

```sas
data one;
  mydate='01jul2018'd;
  put mydate yyqr.;
  put mydate yyqr6.;
  put mydate yyqr7.;
  put mydate yyqr8.;
  put mydate yyqr10.;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put mydate yyqr.;</td>
<td>2018QIII</td>
</tr>
<tr>
<td>put mydate yyqr6.;</td>
<td>18QIII</td>
</tr>
<tr>
<td>put mydate yyqr7.;</td>
<td>2018QIII</td>
</tr>
<tr>
<td>put mydate yyqr8.;</td>
<td>2018QIII</td>
</tr>
<tr>
<td>put mydate yyqr10.;</td>
<td>2018QIII</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “YYQw. Format” on page 438
- “YYQRxw. Format” on page 442

**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
**Syntax**

\texttt{YYQR\textsubscript{x}w.}

**Syntax Description**

\textit{x}

identifies a separator or specifies that no separator appear between the year and the quarter. Here are valid values for \textit{x}.

- \textit{C} separates with a colon.
- \textit{D} separates with a hyphen.
- \textit{N} indicates no separator.
- \textit{P} separates with a period.
- \textit{S} separates with a forward slash.

\textit{w}

specifies the width of the output field.

- **Default**: 8
- **Range**: 6–32

**Interactions**

When \textit{x} is set to \textit{N}, no separator is specified. The width range is 5–32, and the default changes to 7.

When the value of \textit{w} is too small to write a four-digit year, the date appears with only the last two digits of the year.

**Details**

The \texttt{YYQR\textsubscript{x}w.} format writes SAS date values in one of these forms:

- \texttt{<yy>yyqr}
- \texttt{<yy>yyxqr}

The letters and special character are defined as follows:

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

- \textit{x}
  - is a specified separator.

- \texttt{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.

```sas
data _null_
  input date anydtdte12.;
  put date;
  put date yyqrc6.;
  put date yyqrd.;
  put date yyqrn5.;
  put date yyqrp8.;
  put date yyqrsl0.;
datalines;
05/14/2018
;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date;</td>
<td>21318</td>
</tr>
<tr>
<td>put date yyqrc6.;</td>
<td>18:II</td>
</tr>
<tr>
<td>put date yyqrd.;</td>
<td>2018-II</td>
</tr>
<tr>
<td>put date yyqrn5.;</td>
<td>18II</td>
</tr>
<tr>
<td>put date yyqrp8.;</td>
<td>2018.II</td>
</tr>
<tr>
<td>put date yyqrsl0.;</td>
<td>2018/II</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “YYQxw. Format” on page 439
- “YYQRw. Format” on page 441

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}.

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 <\text{yy}> <\text{qq}>. Here is an explanation of the syntax:

<\text{yy}>

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

\( Z \)

specifies that there is no separator.

<\text{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, which is the SAS date value that corresponds to June 29, 2017.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{a=put} (20999,yyqz6.);</td>
<td>201702</td>
</tr>
<tr>
<td>\text{a=put} (20999,yyqz4.);</td>
<td>1702</td>
</tr>
</tbody>
</table>

See Also

- “DS2 Expressions” in *SAS DS2 Programmer’s Guide*

Formats:

- “YYQw. Format” in *SAS DS2 Language Reference*
- “YYQxw. Format” in *SAS DS2 Language Reference*
- “YYQRw. Format” in *SAS DS2 Language Reference*
- “YYQRxw. Format” 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**

YYWEEKUw.

**Syntax Description**

w  
specifies the width of the output field.

**Default** 7

**Range** 2-8

---

**Details**

The YYWEEKUw. format writes a week-number format. The YYWEEKUw. 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:

<table>
<thead>
<tr>
<th>Widths</th>
<th>Formats</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-4</td>
<td>Www</td>
<td>W01</td>
</tr>
<tr>
<td>5-6</td>
<td>yyWww</td>
<td>07W01</td>
</tr>
<tr>
<td>7</td>
<td>yyyyWww</td>
<td>2007W01</td>
</tr>
<tr>
<td>8</td>
<td>yyyy-Www</td>
<td>2007-W01</td>
</tr>
<tr>
<td>9-above</td>
<td>invalid</td>
<td>invalid</td>
</tr>
</tbody>
</table>

**Comparisons**

The YYWEEKUw. format is similar to the WEEKUw. format except that the YYWEEKUw. format does not specify the day-of-week information. Also, the YYWEEKUw. format does not accept any width that is greater than 8.
Example

sasdate = '01JAN2007'd;

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>u=put(sasdate,yyweeku3.);</td>
<td>W00</td>
</tr>
<tr>
<td>v=put(sasdate,yyweeku4.);</td>
<td>W00</td>
</tr>
<tr>
<td>w=put(sasdate,yyweeku5.);</td>
<td>07W00</td>
</tr>
<tr>
<td>x=put(sasdate,yyweeku6.);</td>
<td>07W00</td>
</tr>
<tr>
<td>y=put(sasdate,yyweeku7.);</td>
<td>2007W00</td>
</tr>
<tr>
<td>z=put(sasdate,yyweeku8.);</td>
<td>2007-W00</td>
</tr>
<tr>
<td>put u;</td>
<td></td>
</tr>
<tr>
<td>put v;</td>
<td></td>
</tr>
<tr>
<td>put w;</td>
<td></td>
</tr>
<tr>
<td>put x;</td>
<td></td>
</tr>
<tr>
<td>put y;</td>
<td></td>
</tr>
<tr>
<td>put z;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Format:


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

YYWEEKVw.

Syntax Description

w

specifies the width of the output field.

Default 7

Range 2–8

Details

The YYWEEKVw. 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 zero 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:

<table>
<thead>
<tr>
<th>Widths</th>
<th>Formats</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-4</td>
<td>www</td>
<td>w01</td>
</tr>
<tr>
<td>5-6</td>
<td>yyWww</td>
<td>07W01</td>
</tr>
<tr>
<td>7</td>
<td>yyyyyWww</td>
<td>2007W01</td>
</tr>
<tr>
<td>8</td>
<td>yyyyy-Www</td>
<td>2007-W01</td>
</tr>
<tr>
<td>9-above</td>
<td>invalid</td>
<td>invalid</td>
</tr>
</tbody>
</table>

**Comparisons**

The YYWEEKVw. format is similar to the WEEKVw. format except that the YYWEEKVw. format does not specify the day-of-week information. Also, the YYWEEKVw. format does not accept a width that is greater than 8.

**Example**

```sas
sasdate = '01JAN2007'd;
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>u=put(sasdate,yyweekv3.);</td>
<td>W01</td>
</tr>
<tr>
<td>v=put(sasdate,yyweekv4.);</td>
<td>W01</td>
</tr>
<tr>
<td>w=put(sasdate,yyweekv5.);</td>
<td>07W01</td>
</tr>
<tr>
<td>x=put(sasdate,yyweekv6.);</td>
<td>07W01</td>
</tr>
<tr>
<td>y=put(sasdate,yyweekv7.);</td>
<td>2007W01</td>
</tr>
<tr>
<td>z=put(sasdate,yyweekv8.);</td>
<td>2007-W01</td>
</tr>
<tr>
<td>put u;</td>
<td></td>
</tr>
<tr>
<td>put v;</td>
<td></td>
</tr>
<tr>
<td>put w;</td>
<td></td>
</tr>
<tr>
<td>put x;</td>
<td></td>
</tr>
<tr>
<td>put y;</td>
<td></td>
</tr>
<tr>
<td>put z;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Format:**

**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**

YYWEEKWw.

**Syntax Description**

w

specifies the width of the output field.

- **Default:** 7
- **Range:** 2–8

**Details**

The YYWEEKWw. 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:

<table>
<thead>
<tr>
<th>Widths</th>
<th>Formats</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-4</td>
<td>WWW</td>
<td>W01</td>
</tr>
<tr>
<td>5-6</td>
<td>yyyyWWW</td>
<td>07W01</td>
</tr>
<tr>
<td>7</td>
<td>yyyyWWW</td>
<td>2007W01</td>
</tr>
<tr>
<td>8</td>
<td>yyyy-WWW</td>
<td>2007-W01</td>
</tr>
<tr>
<td>9-above</td>
<td>invalid</td>
<td>invalid</td>
</tr>
</tbody>
</table>

**Comparisons**

The YYWEEKWw. format is similar to the WEEKWw. format except that the YYWEEKWw. format does not specify the day-of-week information. Also, the YYWEEKWw. format does not accept any width that is greater than 8.
Example

sasdate = '01JAN2007'd

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
<td>---------</td>
</tr>
<tr>
<td>u=put(sasdate,yyweekw3.);</td>
<td>W01</td>
</tr>
<tr>
<td>v=put(sasdate,yyweekw4.);</td>
<td>W01</td>
</tr>
<tr>
<td>w=put(sasdate,yyweekw5.);</td>
<td>07W01</td>
</tr>
<tr>
<td>x=put(sasdate,yyweekw6.);</td>
<td>07W01</td>
</tr>
<tr>
<td>y=put(sasdate,yyweekw7.);</td>
<td>2007W01</td>
</tr>
<tr>
<td>z=put(sasdate,yyweekw8.);</td>
<td>2007-W01</td>
</tr>
<tr>
<td>put u;</td>
<td></td>
</tr>
<tr>
<td>put v;</td>
<td></td>
</tr>
<tr>
<td>put w;</td>
<td></td>
</tr>
<tr>
<td>put x;</td>
<td></td>
</tr>
<tr>
<td>put y;</td>
<td></td>
</tr>
<tr>
<td>put z;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

Format:


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= System Option” in SAS System Options: Reference.

Syntax

Zw.d

Syntax Description

w

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1–32</td>
</tr>
</tbody>
</table>
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 \(Zw.d\), \(Zw.d\) 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 BESTw. 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

```plaintext
data one;
    num=1350;
    put num z8.;
run;
```

<table>
<thead>
<tr>
<th>Value of num</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1350</td>
<td>00001350</td>
</tr>
</tbody>
</table>

**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 DECIMALSETTING= 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= System Option” in SAS System Options: Reference.

See: “ZD Format: UNIX” in SAS Companion for UNIX Environments
“ZDw.d Format: Windows” in SAS Companion for Windows
“ZDw.d Format: z/OS” in SAS Companion for z/OS

Syntax

\texttt{ZDw.d}

Syntax Description

\textit{w} 

specifies the width of the output field.

Default 1

Range 1–32

\textit{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.

\textit{Note}: Different operating environments store zoned decimal values in different ways. However, the ZDw.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:

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

data one;
  num=1350;
  put num zd8.;
run;

<table>
<thead>
<tr>
<th>Value of num</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1350</td>
<td>0000135 {</td>
</tr>
</tbody>
</table>

* The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each byte occupies one column of the output field.
Part 2

SAS Informats

Chapter 3

About Informats .......................................................... 457

Chapter 4

Dictionary of Informats .................................................. 475
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:

\[<$\text{informat}\text{<w>},\text{<d}>}$\]

$\quad$ indicates a character informat; its absence indicates a numeric informat.

\textit{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 \textit{Base SAS Procedures Guide}.

\textit{w}

specifies the informat width, which for most informats is the number of columns in the input data.

\textit{d}

specifies an optional decimal scaling factor in the numeric informats. SAS divides the input data by 10 to the power of \textit{d}.

\textit{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 \textit{w} and \textit{d} values from the informat, SAS uses default values. If the data contains decimal points, SAS ignores the \textit{d} value and reads the number of decimal places that are actually in the input data.

If the informat width is too narrow to read all the columns in the input data, you might get unexpected results. The problem frequently occurs with the date and time informats. You must adjust the width of the informat to include blanks or special characters between the day, month, year, or time. For more information about date and time values, see “Dates, Times, and Intervals” in \textit{SAS Language Reference: Concepts}.

When a problem occurs with an informat, SAS writes a note to the SAS log and assigns a missing value to the variable. Problems occur if you use an incompatible informat such as a numeric informat to read character data, or if you specify the width of a date and time informat that causes SAS to read a special character in the last column.

Using Informats

Ways to Specify Informats

\textbf{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:

```sas
input @15 style $3. @21 price 5.2;
```

The `@w.` character informat reads values into the variable `STYLE`. The `w.d` numeric informat reads values into the variable `PRICE`.

For a complete discussion of the INPUT statement, see “INPUT Statement” in *SAS 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:

```sas
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 Function” in *SAS Functions and CALL Routines: Reference*. For a complete discussion of the INPUT function, see “INPUT Function” in *SAS Functions and CALL Routines: Reference*.

**INFORMAT Statement**
The INFORMAT statement associates an informat with a variable. SAS uses the informat in any subsequent INPUT statement to read values into the variable. For example, in the following statements, the INFORMAT statement associates the DATE informat with the variables Birthdate and Interview:

```sas
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 Statement, 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 Statement, 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 DATEw. informat with the variables Birthdate and Interview:

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

An informat that is associated by using the INFORMAT= option in the ATTRIB statement behaves like an informat that you specify with a colon (:) format modifier in an INPUT statement. For details about using the colon (:) modifier, see “INPUT Statement, List” in SAS 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 Statement” 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:

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

Although using NOFMTERR enables SAS to process a variable, you lose the information that the user-defined informat supplies. This option can cause a DATA step to misread data, and it can produce incorrect results. For more information, see “FMTERR System Option” in SAS System Options: Reference.

To avoid problems, make sure that users of your program have access to all the user-defined informats that are used.

### Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms

#### Definitions

Integer values for integer binary data are typically stored in one of three sizes: 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.

<table>
<thead>
<tr>
<th>Platform for Which the Data Was Created</th>
<th>Platform the Data Is Read On</th>
<th>Signed Integer</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>big endian</td>
<td>big endian</td>
<td>yes</td>
<td>IB or S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>big endian</td>
<td>no</td>
<td>PIB, S370FPIB, S370FIBU</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>no</td>
<td>PIBR</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>no</td>
<td>PIBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>yes</td>
<td>IB or IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>no</td>
<td>PIB or PIBR</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>yes</td>
<td>S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>no</td>
<td>S370FPIB</td>
</tr>
<tr>
<td>little endian</td>
<td>either</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>either</td>
<td>no</td>
<td>PIBR</td>
</tr>
</tbody>
</table>

**Integer Binary Notation in Different Programming Languages**

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

<table>
<thead>
<tr>
<th>IBM VS COBOL II Clauses</th>
<th>Corresponding S370Fxxx Formats and Informats</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC S9(X) PACKED-DECIMAL</td>
<td>S370FPDw.</td>
</tr>
<tr>
<td>PIC 9(X) PACKED-DECIMAL</td>
<td>S370FPDUw.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY</td>
<td>S370FZDw.</td>
</tr>
<tr>
<td>PIC 9(W) DISPLAY</td>
<td>S370FZDUw.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN LEADING</td>
<td>S370FZDLw.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN LEADING SEPARATE</td>
<td>S370FZDSw.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN TRAILING SEPARATE</td>
<td>S370FZDTw.</td>
</tr>
</tbody>
</table>

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 $\text{ceil}\left(\frac{x+1}{2}\right)$. For PIC 9(X) PACKED-DECIMAL, W is $\text{ceil}\left(\frac{x}{2}\right)$. For example, PIC S9(5) PACKED-DECIMAL represents five digits. If a sign is included, six nibbles are needed. $\text{ceil}\left(\frac{5+1}{2}\right)$ 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:

<table>
<thead>
<tr>
<th>offset</th>
<th>value (in hex)</th>
<th>inst label</th>
<th>directive</th>
</tr>
</thead>
<tbody>
<tr>
<td>+000000 00001C</td>
<td>2 PEX1</td>
<td>DC PL3'1'</td>
<td></td>
</tr>
<tr>
<td>+000003 00001D</td>
<td>3 PEX2</td>
<td>DC PL3'-1'</td>
<td></td>
</tr>
<tr>
<td>+000006 00F0C1</td>
<td>4 ZEX1</td>
<td>DC ZL3'1'</td>
<td></td>
</tr>
<tr>
<td>+000009 00F0D1</td>
<td>5 ZEX2</td>
<td>DC ZL3'1'</td>
<td></td>
</tr>
</tbody>
</table>

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 Informs**

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

<table>
<thead>
<tr>
<th>Format</th>
<th>Data Type Representation</th>
<th>Corresponding Informat</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PD</td>
<td>Packed decimal</td>
<td>PD</td>
<td>Local signed packed decimal.</td>
</tr>
<tr>
<td>PK</td>
<td>Packed decimal</td>
<td>PK</td>
<td>Unsigned packed decimal; not specific to your operating environment.</td>
</tr>
<tr>
<td>ZD</td>
<td>Zoned decimal</td>
<td>ZD</td>
<td>Local zoned decimal.</td>
</tr>
<tr>
<td>none</td>
<td>Zoned decimal</td>
<td>ZDB</td>
<td>Translates EBCDIC blank (x'40') to EBCDIC zero (x'F0'), and then corresponds to the informat as zoned decimal.</td>
</tr>
<tr>
<td>none</td>
<td>Zoned decimal</td>
<td>ZDV</td>
<td>Zoned decimal representation other than IBM.</td>
</tr>
<tr>
<td>S370FPD</td>
<td>Packed decimal</td>
<td>S370FPD</td>
<td>Last nibble C (positive) or D (negative).</td>
</tr>
<tr>
<td>S370FPDU</td>
<td>Packed decimal</td>
<td>S370FPDU</td>
<td>Last nibble always F (positive).</td>
</tr>
<tr>
<td>S370FZD</td>
<td>Zoned decimal</td>
<td>S370FZD</td>
<td>Last byte contains sign in upper nibble: C (positive) or D (negative).</td>
</tr>
<tr>
<td>S370FZDU</td>
<td>Zoned decimal</td>
<td>S370FZDU</td>
<td>Unsigned; sign nibble always F.</td>
</tr>
<tr>
<td>S370FZDL</td>
<td>Zoned decimal</td>
<td>S370FZDL</td>
<td>Sign nibble in first byte in informat; separate leading sign byte of x'C0' (positive) or x'D0' (negative) in format.</td>
</tr>
<tr>
<td>S370FZDS</td>
<td>Zoned decimal</td>
<td>S370FZDS</td>
<td>Leading sign of – (x'60') or + (x'4E').</td>
</tr>
<tr>
<td>S370FZDT</td>
<td>Zoned decimal</td>
<td>S370FZDT</td>
<td>Trailing sign of – (x'60') or + (x'4E').</td>
</tr>
<tr>
<td>Format</td>
<td>Data Type Representation</td>
<td>Corresponding Informat</td>
<td>Comment</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------</td>
<td>------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PDJULI</td>
<td>Packed decimal</td>
<td>PDJULI</td>
<td>Julian date in packed representation - IBM computation</td>
</tr>
<tr>
<td>PDJULG</td>
<td>Packed decimal</td>
<td>PDJULG</td>
<td>Julian date in packed representation - Gregorian computation</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>RMFDUR</td>
<td>Input layout is: mmssstttF</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>SHRSTAMP</td>
<td>Input layout is yyyydddFhhmmstth, where yyyydddF is the packed Julian date; yyyy is a zero-based year from 1900.</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>SMFSTAMP</td>
<td>Input layout is xxxxxxxxxxxyydddF, where yyyydddF is the packed Julian date; yyyy is a zero-based year from 1900.</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>PDTIME</td>
<td>Input layout is 0hhmmssF.</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>RMFSTAMP</td>
<td>Input layout is 0hhmmssFyyyydddF, where yyyydddF is the packed Julian date; yyyy is a zero-based year from 1900.</td>
</tr>
</tbody>
</table>

**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:
<table>
<thead>
<tr>
<th>Date, Time, or Datetime</th>
<th>ISO 8601 Notation</th>
<th>Example</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>YYYYMMDD</td>
<td>20180915</td>
<td>B8601DAw.</td>
</tr>
<tr>
<td>Time with no time zone offset</td>
<td>hhmmssssssss</td>
<td>155300322348</td>
<td>B8601TMw.d</td>
</tr>
<tr>
<td>Time with a time zone offset</td>
<td>hhmmss+-hhmm</td>
<td>155300-0500</td>
<td>B8601TZw.d</td>
</tr>
<tr>
<td>hhmmssZ</td>
<td>155300Z</td>
<td>B8601TZw.d</td>
<td></td>
</tr>
<tr>
<td>Convert to local time with a time zone offset</td>
<td>hhmmss+-hhmm</td>
<td>155300+0500</td>
<td>B8601TZw.d</td>
</tr>
<tr>
<td>Datetime with no time zone offset</td>
<td>yyyyymmddThhmmssssss</td>
<td>20180915T155300</td>
<td>B8601DTw.d</td>
</tr>
<tr>
<td>Datetime with a time zone offset</td>
<td>yyyyymmddThhmmss+-hhmm</td>
<td>20180915T155300+0500</td>
<td>B8601DZw.d</td>
</tr>
<tr>
<td>yyyyymmddThhmmssZ</td>
<td>20180915T155300Z</td>
<td>B8601DZw.d</td>
<td></td>
</tr>
<tr>
<td>Date from a datetime and no time zone offset</td>
<td>yyyyymmdd</td>
<td>20180915</td>
<td>B8601DNw.</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Date, Time, or Datetime</th>
<th>ISO 8601 Notation</th>
<th>Example</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>yyyy-mm-dd</td>
<td>2018-09-15</td>
<td>E8601DAw.</td>
</tr>
<tr>
<td>Time with no time zone offset</td>
<td>hh:mm:ss.nnnnnn</td>
<td>15:53:00.322348</td>
<td>E8601TMw.d</td>
</tr>
<tr>
<td>Time with a time zone offset</td>
<td>hh:mm:ss.nnnnnn+-hh:mm</td>
<td>15:53:00+05:00</td>
<td>E8601TZw.d</td>
</tr>
<tr>
<td>Convert to local time with a time zone offset</td>
<td>hh:mm:ss.nnnnnn+-hh:mm</td>
<td>15:53:00+05:00</td>
<td>E8601LZw.d</td>
</tr>
<tr>
<td>Datetime with no time zone offset</td>
<td>yyyy-mm-ddThh:mm:ss.nnnnnn</td>
<td>2018-09-15T15:53:00</td>
<td>E8601DTw.d</td>
</tr>
<tr>
<td>Datetime with a time zone offset</td>
<td>yyyy-mm-ddThh:mm:ss.nnnnnn+-hh:mm</td>
<td>201809-15T15:53:00+05:00</td>
<td>E8601DZw.d</td>
</tr>
</tbody>
</table>
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 45.

Reading ISO 8601 Duration, Interval, and Datetime Values

Informs That Read Duration, Interval, and Datetime Values

SAS uses two informs 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 informs result in a SAS character representation. If you want a datetime value to be read as a numeric value, use the B8601DT informat, the B8601DZ informat, the E8601DT informat, or the E8601DZ informat.

Complete Duration, Interval, and Datetime Notations

The following table shows the formatting of duration, datetime, and interval values that can be read in the complete form:

<table>
<thead>
<tr>
<th>Time Component</th>
<th>ISO 8601 Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration - Basic Notation</td>
<td>PYYYYMMDDThhmmss</td>
<td>P20180915T155300</td>
</tr>
<tr>
<td></td>
<td>–PYYYYMMDDThhmmss</td>
<td>–P20180915T155300</td>
</tr>
<tr>
<td>Duration - Extended Notation</td>
<td>PYYYY-MM-DDThh:mm:ss</td>
<td>P2018-09-15T15:53:00</td>
</tr>
</tbody>
</table>
### 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 `Pyyyyymmd`. 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:

- **Duration:**
  - `p0003-02-T10:31:33`
    - The omitted component is the number of days.
  - `–p0003-02-T:31:33`
    - The omitted component is the number of hours.

- **Datetime:**
  - `YYYY-MM-DDThh:mm:ss.fff` + `|` `-hhmm`
    - `20180915T15:53:00` + `04:30`
    - `20180915T15:53:00`

### Time Component | ISO 8601 Notation | Example
--- | --- | ---
Duration - Basic and Extended Notation | `–PYYYY-MM-DDThh:mm:ss` | `–P2018-09-15T15:53:00`
| | `PNyNnDTnHNnMS` | `P2y10m14dT20h13m45s`
| | `–PnYNnDTnHNnMS` | `–P2n10m14dT20h13m45s`
| | `pNw` | `P6w`
Interval - Basic Notation | `YYYYyMMDDThhmmss/YYYMDDDDThhhms` | `20180915T155300/20141113T000000`
| | `PnYNnDTnHNnMS/YYYMDDDDThhhms` | `P2y10M14dT20h13m45s/20180915T155300`
| | `YYYMDDDDThhmmss/PnYNnDTnHNnMS` | `20180915T155300/P2y10M14dT20h13m45s`
Interval - Extended Notation | `YYYY-MM-DDThh:mm:ss/YYYY-MM-DDThh:mm:ss` | `2018-09-15T15:53:00/2014-11-13T00:00:00`
| | `PnYNnDTnHNnMS/YYYY-MM-DDThh:mm:ss` | `P2y10M14dT20h13m45s/2018-09-15T15:53:00`
| | `YYYY-MM-DDThh:mm:ss/PnYNnDTnHNnMS` | `2018-09-15T15:53:00/P2y10M14dT20h13m45s`
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`
- `-p0003-03T--:-:-`
- `P2y3m4dT5h6m`
- `2018-09-Tx: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:

<table>
<thead>
<tr>
<th>Duration</th>
<th>Extended Normalized Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>p3y13m</td>
<td>p0004-01</td>
</tr>
<tr>
<td>pt24h24m65s</td>
<td>P----:01T:25:05</td>
</tr>
<tr>
<td>p3y13mT24h61m</td>
<td>P0004-01-01T01:01</td>
</tr>
<tr>
<td>p0004-13</td>
<td>p0005-01</td>
</tr>
<tr>
<td>p0003-02-61T15:61</td>
<td>P0003-04-01T16:02:01</td>
</tr>
<tr>
<td>pt13m</td>
<td>P13M</td>
</tr>
</tbody>
</table>

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
Chapter 4
Dictionary of Informats

Informats Documented in Other Publications .......................... 477
Informats by Category ....................................................... 477
Dictionary ................................................................. 488
$ASCIIw. Informat ....................................................... 488
$BASE64Xw. Informat ................................................... 489
$BINARYWw. Informat .................................................... 490
$CBw. Informat .......................................................... 492
$CHARw. Informat ........................................................ 493
$CHARZBw. Informat ..................................................... 494
$EBCDICw. Informat ..................................................... 495
$HEXw. Informat .......................................................... 496
$N8601Bw.d Informat .................................................... 497
$N8601Ew.d Informat ..................................................... 499
$OCTALw. Informat ....................................................... 502
$PHExw. Informat ........................................................ 503
$QUOTEw. Informat ....................................................... 504
$UPCASEw. Informat ..................................................... 504
$UUIDw. Informat ....................................................... 505
$VARYINGw. Informat .................................................... 506
$w. Informat .............................................................. 508
ANYDTDTEw. Informat .................................................. 509
ANYDTDTMw. Informat .................................................. 512
ANYD'T'TMEw. Informat .................................................. 515
B8601CIw.d Informat ..................................................... 517
B8601DAw. Informat ...................................................... 518
B8601DJw.d Informat ..................................................... 520
B8601DNw. Informat ...................................................... 521
B8601DTw.d Informat .................................................... 523
B8601DXw. Informat ...................................................... 524
B8601DZw.d Informat .................................................... 525
B8601LXw. Informat ...................................................... 526
B8601MW. d Informat ..................................................... 527
B8601TXw. Informat ...................................................... 528
B8601TZWw.d Informat ................................................... 529
BINARYw.d Informat ..................................................... 530
BITSw.d Informat ........................................................ 531
BZw.d Informat .......................................................... 532
CBw.d Informat .......................................................... 533
COMMw.d Informat ....................................................... 535
COMMAXw.d Informat .................................................... 536
DATEw. Informat .............................. 537
DATETIMEw. Informat .......................... 538
DDMMYYw. Informat .............................. 540
E8601DAw. Informat ............................... 542
E8601DNw. Informat .............................. 543
E8601DTw.d Informat .............................. 544
E8601DXw. Informat .............................. 546
E8601DZw.d Informat .............................. 546
E8601LXw. Informat .............................. 548
E8601LZw.d Informat .............................. 548
E8601TMw.d Informat .............................. 550
E8601TXw. Informat .............................. 552
E8601TZw.d Informat .............................. 552
FLOATw.d Informat .............................. 554
HEXw. Informat ................................. 555
HHMMSSw. Informat .............................. 556
IBw.d Informat ................................. 558
IBRw.d Informat ................................. 559
IEEEw.d Informat .............................. 561
JULIANw. Informat .............................. 562
MDYAMPMw.d Informat .......................... 563
MMDDYYw. Informat .............................. 565
MONYYw. Informat .............................. 567
MSECw. Informat ............................... 568
NUMXw.d Informat .............................. 569
OCTALw.d Informat .............................. 571
PDw.d Informat ................................. 572
PDJULGw. Informat .............................. 573
PDJULw. Informat ............................... 575
PDTIMEw. Informat .............................. 576
PERCENTw.d Informat ............................ 577
PIBw.d Informat ............................... 578
PIBRw.d Informat ............................... 580
PKw.d Informat ................................. 582
PUNCH.d Informat ............................... 583
RBw.d Informat ................................. 584
RMFDURw. Informat ............................. 585
RMFSTAMPw. Informat .......................... 587
ROWw.d Informat ............................... 588
S370FFw.d Informat ............................. 590
S370FIBw.d Informat ............................. 591
S370FIBUw.d Informat .......................... 592
S370FPDw.d Informat ............................ 594
S370FPDUw.d Informat .......................... 595
S370FPICw.d Informat ........................... 596
S370FRBw.d Informat ............................ 598
S370FZDw.d Informat ............................ 599
S370FZDBw.d Informat ........................... 600
S370FZDLw.d Informat ........................... 601
S370FZDSw.d Informat ........................... 603
S370FZDTw.d Informat ........................... 604
S370FZDUw.d Informat ........................... 605
SHRRSTAMPw. Informat ........................ 606
SMFSTAMPw. Informat ........................... 607
STIMERw. Informat .............................. 609
TIMEw. Informat ............................... 610
Informats Documented in Other Publications


Informats by Category

There are seven categories of informats in this list:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS</td>
<td>instructs SAS to read character data values into character variables in CAS.</td>
</tr>
<tr>
<td>Character</td>
<td>instructs SAS to read character data values into character variables.</td>
</tr>
<tr>
<td>Column Binary</td>
<td>instructs SAS to read data stored in column-binary or multipunched form into character or numeric variables.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>instructs SAS to read date values into variables that represent dates, times, and datetimes.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>instructs SAS to read date, time, and datetime values that are written in the ISO 8601 standard into either numeric or character variables.</td>
</tr>
<tr>
<td>Numeric</td>
<td>instructs SAS to read numeric data values into numeric variables.</td>
</tr>
</tbody>
</table>

For information about column-binary data, see “Reading Column-Binary Data” in SAS Language Reference: Concepts. 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.

<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS</td>
<td>$ASCIIw. Informat (p. 488)</td>
<td>Converts ASCII character data to native format.</td>
</tr>
<tr>
<td></td>
<td>$BASE64Xw. Informat (p. 489)</td>
<td>Converts ASCII text to character data by using Base 64 encoding.</td>
</tr>
<tr>
<td></td>
<td>$BINARYw. Informat (p. 490)</td>
<td>Converts binary data to character data.</td>
</tr>
<tr>
<td></td>
<td>$CBw. Informat (p. 492)</td>
<td>Reads standard character data from column-binary files.</td>
</tr>
<tr>
<td></td>
<td>$CHARw. Informat (p. 493)</td>
<td>Reads character data with blanks.</td>
</tr>
<tr>
<td></td>
<td>$CHARZBw. Informat (p. 494)</td>
<td>Converts binary 0s to blanks.</td>
</tr>
<tr>
<td></td>
<td>$EBCDICw. Informat (p. 495)</td>
<td>Converts EBCDIC character data to native format.</td>
</tr>
<tr>
<td></td>
<td>$HEXw. Informat (p. 496)</td>
<td>Converts hexadecimal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$N8601Bw.d Informat (p. 497)</td>
<td>Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notation.</td>
</tr>
<tr>
<td></td>
<td>$N8601Ew.d Informat (p. 499)</td>
<td>Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.</td>
</tr>
<tr>
<td></td>
<td>$OCTALw. Informat (p. 502)</td>
<td>Converts octal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$PHEXw. Informat (p. 503)</td>
<td>Converts packed hexadecimal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$QUOTEw. Informat (p. 504)</td>
<td>Removes matching quotation marks from character data.</td>
</tr>
<tr>
<td></td>
<td>$UPCASEw. Informat (p. 504)</td>
<td>Converts character data to uppercase.</td>
</tr>
<tr>
<td></td>
<td>$VARYINGw. Informat (p. 506)</td>
<td>Reads character data of varying length.</td>
</tr>
<tr>
<td></td>
<td>$w. Informat (p. 508)</td>
<td>Reads standard character data.</td>
</tr>
<tr>
<td></td>
<td>ANYYTDTEw. Informat (p. 509)</td>
<td>Reads and extracts the date value from various date, time, and datetime forms.</td>
</tr>
<tr>
<td></td>
<td>ANYYTDTDMw. Informat (p. 512)</td>
<td>Reads and extracts datetime values from various date, time, and datetime forms.</td>
</tr>
<tr>
<td></td>
<td>ANYYDTTMEw. Informat (p. 515)</td>
<td>Reads and extracts time values from various date, time, and datetime forms.</td>
</tr>
<tr>
<td></td>
<td>BINARYw.d Informat (p. 530)</td>
<td>Converts positive binary values to integers.</td>
</tr>
<tr>
<td></td>
<td>BITSw.d Informat (p. 531)</td>
<td>Extracts bits.</td>
</tr>
<tr>
<td></td>
<td>BZw.d Informat (p. 532)</td>
<td>Converts blanks to 0s.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CBw.d</td>
<td>Informat (p. 533)</td>
<td>Reads standard numeric values from column-binary files.</td>
</tr>
<tr>
<td>COMMAw.d</td>
<td>Informat (p. 535)</td>
<td>Removes embedded characters.</td>
</tr>
<tr>
<td>COMMAXw.d</td>
<td>Informat (p. 536)</td>
<td>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.</td>
</tr>
<tr>
<td>DATEw.</td>
<td>Informat (p. 537)</td>
<td>Reads date values in the form ddmmyy or ddmmyyyyy.</td>
</tr>
<tr>
<td>DATETIMEw.</td>
<td>Informat (p. 538)</td>
<td>Reads datetime values in the form ddmmyy hh:mm:ss or ddmmyyyyy hh:mm:ss.ss.</td>
</tr>
<tr>
<td>DDMMYYw.</td>
<td>Informat (p. 540)</td>
<td>Reads date values in the form ddmmyy&lt;yy&gt; or dd-mm-yy&lt;yy&gt;, 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.</td>
</tr>
<tr>
<td>E8601DAw.</td>
<td>Informat (p. 542)</td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DNw.</td>
<td>Informat (p. 543)</td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd and returns SAS datetime values where the time portion of the value is 000000.</td>
</tr>
<tr>
<td>E8601DTw.d</td>
<td>Informat (p. 544)</td>
<td>Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.&lt;fffff&gt;.</td>
</tr>
<tr>
<td>E8601DXw.</td>
<td>Informat (p. 546)</td>
<td>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-ddThh:mm:ss±hh:mm.</td>
</tr>
<tr>
<td>E8601DZw.d</td>
<td>Informat (p. 546)</td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime extended notation yyyy-mm-ddThh:mm:ss±hh:mm.&lt;fffff&gt; or yyyy-mm-ddThh:mm:ss.&lt;fffff&gt;Z.</td>
</tr>
<tr>
<td>E8601TMw.d</td>
<td>Informat (p. 550)</td>
<td>Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.&lt;fffff&gt;.</td>
</tr>
<tr>
<td>E8601TZw.d</td>
<td>Informat (p. 552)</td>
<td>Reads time values that are specified using the ISO 8601 extended time notation hh:mm:ss±hh:mm.&lt;fffff&gt; or hh:mm:ssZ.</td>
</tr>
<tr>
<td>FLOATw.d</td>
<td>Informat (p. 554)</td>
<td>Reads a native single-precision, floating-point value and divides it by 10 raised to the dth power.</td>
</tr>
<tr>
<td>HEXw.</td>
<td>Informat (p. 555)</td>
<td>Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values.</td>
</tr>
<tr>
<td>HHMNSSw.</td>
<td>Informat (p. 556)</td>
<td>Reads hours, minutes, and seconds in the form hh:mm:ss or hh:mm:ss.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>IBw.d Informat (p. 558)</td>
<td>Reads native integer binary (fixed-point) values, including negative values.</td>
<td></td>
</tr>
<tr>
<td>IBRw.d Informat (p. 559)</td>
<td>Reads integer binary (fixed-point) values in Intel and DEC formats.</td>
<td></td>
</tr>
<tr>
<td>IEEEw.d Informat (p. 561)</td>
<td>Reads an IEEE floating-point value and divides it by 10 raised to the $d$th power.</td>
<td></td>
</tr>
<tr>
<td>JULIANw. Informat (p. 562)</td>
<td>Reads Julian dates in the form $yyddd$ or $yyyyddd$.</td>
<td></td>
</tr>
<tr>
<td>MDYAMPMw.d Informat (p. 563)</td>
<td>Reads datetime values in the form $mm-dd-yy&lt;yy&gt; hh:mm:ss.ss \ AM</td>
<td>PM$, 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.</td>
</tr>
<tr>
<td>MMDDYYw. Informat (p. 565)</td>
<td>Reads date values in the form $mmddyy$ or $mmddyyyy$.</td>
<td></td>
</tr>
<tr>
<td>MONYYw. Informat (p. 567)</td>
<td>Reads month and year date values in the form $mmm yy$ or $mmmyyyy$.</td>
<td></td>
</tr>
<tr>
<td>MSECw. Informat (p. 568)</td>
<td>Reads TIME MIC values.</td>
<td></td>
</tr>
<tr>
<td>NUMXw.d Informat (p. 569)</td>
<td>Reads numeric values with a comma in place of the decimal point.</td>
<td></td>
</tr>
<tr>
<td>PDw.d Informat (p. 572)</td>
<td>Reads data that is stored in IBM packed decimal format.</td>
<td></td>
</tr>
<tr>
<td>PDTIMEw. Informat (p. 576)</td>
<td>Reads packed decimal time of SMF and RMF records.</td>
<td></td>
</tr>
<tr>
<td>PERCENTw.d Informat (p. 577)</td>
<td>Reads percentages as numeric values.</td>
<td></td>
</tr>
<tr>
<td>PIBw.d Informat (p. 578)</td>
<td>Reads positive integer binary (fixed-point) values.</td>
<td></td>
</tr>
<tr>
<td>PIBRw.d Informat (p. 580)</td>
<td>Reads positive integer binary (fixed-point) values in Intel and DEC formats.</td>
<td></td>
</tr>
<tr>
<td>PKw.d Informat (p. 582)</td>
<td>Reads unsigned packed decimal data.</td>
<td></td>
</tr>
<tr>
<td>PUNCH.d Informat (p. 583)</td>
<td>Reads whether a row of column-binary data is punched.</td>
<td></td>
</tr>
<tr>
<td>RBw.d Informat (p. 584)</td>
<td>Reads numeric data that is stored in real binary (floating-point) notation.</td>
<td></td>
</tr>
<tr>
<td>RMFDURw. Informat (p. 585)</td>
<td>Reads duration intervals of RMF records.</td>
<td></td>
</tr>
<tr>
<td>RMFSTAMPw. Informat (p. 587)</td>
<td>Reads time and date fields of RMF records.</td>
<td></td>
</tr>
<tr>
<td>S370FFw.d Informat (p. 590)</td>
<td>Reads EBCDIC numeric data.</td>
<td></td>
</tr>
<tr>
<td>S370FIBw.d Informat (p. 591)</td>
<td>Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>S370FIBU</td>
<td>w.d Informat</td>
<td>Reads unsigned integer binary (fixed-point) values in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPD</td>
<td>w.d Informat</td>
<td>Reads packed data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPDU</td>
<td>w.d Informat</td>
<td>Reads unsigned packed decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPIB</td>
<td>w.d Informat</td>
<td>Reads positive integer binary (fixed-point) values in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FRB</td>
<td>w.d Informat</td>
<td>Reads real binary (floating-point) data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZD</td>
<td>w.d Informat</td>
<td>Reads zoned decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDB</td>
<td>w.d Informat</td>
<td>Reads zoned decimal data in which 0s have been left blank.</td>
</tr>
<tr>
<td>S370FZDL</td>
<td>w.d Informat</td>
<td>Reads zoned decimal leading-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDS</td>
<td>w.d Informat</td>
<td>Reads zoned decimal, separate leading-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDT</td>
<td>w.d Informat</td>
<td>Reads zoned decimal, separate trailing-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDU</td>
<td>w.d Informat</td>
<td>Reads unsigned zoned decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>SHRSTAMP</td>
<td>w. Informat</td>
<td>Reads date and time values of SHR records.</td>
</tr>
<tr>
<td>SMFSTAMP</td>
<td>w. Informat</td>
<td>Reads time and date values of SMF records.</td>
</tr>
<tr>
<td>TIME</td>
<td>w. Informat</td>
<td>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.</td>
</tr>
<tr>
<td>TODSTAMP</td>
<td>w. Informat</td>
<td>Reads an 8-byte time-of-day stamp.</td>
</tr>
<tr>
<td>TRAILSGN</td>
<td>w. Informat</td>
<td>Reads a trailing plus (+) or minus (−) sign.</td>
</tr>
<tr>
<td>TU</td>
<td>w. Informat</td>
<td>Reads timer units.</td>
</tr>
<tr>
<td>VAXRB</td>
<td>w.d Informat</td>
<td>Reads real binary (floating-point) data in VMS format.</td>
</tr>
<tr>
<td>VMSZN</td>
<td>w.d Informat</td>
<td>Reads VMS and MicroFocus COBOL zoned numeric data.</td>
</tr>
<tr>
<td>w.d Informat</td>
<td></td>
<td>Reads standard numeric data.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>YYMDDw. Informat</td>
<td>Reads date values in the form yymmdd or yyyyymmdd.</td>
</tr>
<tr>
<td></td>
<td>YYMMNw. Informat</td>
<td>Reads date values in the form yyyyymm or yymm.</td>
</tr>
<tr>
<td></td>
<td>YYQw. Informat</td>
<td>Reads quarters of the year in the form yyyyQq or yyyyQq.</td>
</tr>
<tr>
<td></td>
<td>ZDw.d Informat</td>
<td>Reads zoned decimal data.</td>
</tr>
<tr>
<td></td>
<td>ZDBw.d Informat</td>
<td>Reads zoned decimal data in which zeros have been left blank.</td>
</tr>
<tr>
<td></td>
<td>ZDVw.d Informat</td>
<td>Reads and validates zoned decimal data.</td>
</tr>
<tr>
<td></td>
<td>$ASCIIw. Informat</td>
<td>Converts ASCII character data to native format.</td>
</tr>
<tr>
<td></td>
<td>$BASE64Xw. Informat</td>
<td>Converts ASCII text to character data by using Base 64 encoding.</td>
</tr>
<tr>
<td></td>
<td>$BINARYw. Informat</td>
<td>Converts binary data to character data.</td>
</tr>
<tr>
<td></td>
<td>$CHARw. Informat</td>
<td>Reads character data with blanks.</td>
</tr>
<tr>
<td></td>
<td>$CHARZBw. Informat</td>
<td>Converts binary 0s to blanks.</td>
</tr>
<tr>
<td></td>
<td>$EBCDICw. Informat</td>
<td>Converts EBCDIC character data to native format.</td>
</tr>
<tr>
<td></td>
<td>$HEXw. Informat</td>
<td>Converts hexadecimal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$OCTALw. Informat</td>
<td>Converts octal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$PHEXw. Informat</td>
<td>Converts packed hexadecimal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$QUOTEw. Informat</td>
<td>Removes matching quotation marks from character data.</td>
</tr>
<tr>
<td></td>
<td>$UPCASEw. Informat</td>
<td>Converts character data to uppercase.</td>
</tr>
<tr>
<td></td>
<td>$VARYINGw. Informat</td>
<td>Reads character data of varying length.</td>
</tr>
<tr>
<td></td>
<td>$w. Informat</td>
<td>Reads standard character data.</td>
</tr>
<tr>
<td></td>
<td>SCBw. Informat</td>
<td>Reads standard character data from column-binary files.</td>
</tr>
<tr>
<td></td>
<td>CBw.d Informat</td>
<td>Reads standard numeric values from column-binary files.</td>
</tr>
<tr>
<td></td>
<td>PUNCH.d Informat</td>
<td>Reads whether a row of column-binary data is punched.</td>
</tr>
<tr>
<td></td>
<td>ROWw.d Informat</td>
<td>Reads a column-binary field down a card column.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>SN8601Bw.d Informat</td>
<td>Reads complete, truncated, and omitted forms of ISO 8601 duration, datetme, and interval values that are specified in either the basic or extended notation.</td>
</tr>
<tr>
<td></td>
<td>SN8601Ew.d Informat</td>
<td>Reads ISO 8601 duration, datetme, and interval values that are specified in the extended notation.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>ANYDTEw.</td>
<td>Informat (p. 509)</td>
<td>Reads and extracts the date value from various date, time, and datetime forms.</td>
</tr>
<tr>
<td>ANYDTDMw.</td>
<td>Informat (p. 512)</td>
<td>Reads and extracts datetime values from various date, time, and datetime forms.</td>
</tr>
<tr>
<td>ANYDTTMEw.</td>
<td>Informat (p. 515)</td>
<td>Reads and extracts time values from various date, time, and datetime forms.</td>
</tr>
<tr>
<td>B8601CIw.d</td>
<td>Informat (p. 517)</td>
<td>Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss&lt;fff&gt;.</td>
</tr>
<tr>
<td>B8601DAw.</td>
<td>Informat (p. 518)</td>
<td>Reads date values that are specified using the ISO 8601 base notation yyyymmdd.</td>
</tr>
<tr>
<td>B8601DJw.d</td>
<td>Informat (p. 520)</td>
<td>Reads a Java date and time value that is in the form yyyymmddhhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>B8601DNw.</td>
<td>Informat (p. 521)</td>
<td>Reads date values that are specified using the ISO 8601 basic notation yyyymmdd and returns SAS datetime values where the time portion of the value is 000000.</td>
</tr>
<tr>
<td>B8601DTw.d</td>
<td>Informat (p. 523)</td>
<td>Reads datetime values that are specified using the ISO 8601 basic notation yyyymmddThhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>B8601DZw.d</td>
<td>Informat (p. 525)</td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation yyyymmddThhmm:ss±hhmm or yyyymmddThhmmss&lt;ffffff&gt;Z.</td>
</tr>
<tr>
<td>B8601TMw.d</td>
<td>Informat (p. 527)</td>
<td>Reads time values that are specified using the ISO 8601 basic notation hhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>B8601TZw.d</td>
<td>Informat (p. 528)</td>
<td>Reads time values that are specified using the ISO 8601 basic time notation hhmmss&lt;ffffff&gt;+</td>
</tr>
<tr>
<td>DATEw.</td>
<td>Informat (p. 537)</td>
<td>Reads date values in the form ddmmyy or ddmmyyyy.</td>
</tr>
<tr>
<td>DATETIMEw.</td>
<td>Informat (p. 538)</td>
<td>Reads datetime values in the form ddmmyy hh:mm:ss ss or ddmmyyyyy hh:mm:ss ss.</td>
</tr>
<tr>
<td>DDMMYYw.</td>
<td>Informat (p. 540)</td>
<td>Reads date values in the form ddmmyy&lt;yy&gt; or dd-mm-yyyy&lt;yy&gt;, 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.</td>
</tr>
<tr>
<td>E8601DAw.</td>
<td>Informat (p. 542)</td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DNw.</td>
<td>Informat (p. 543)</td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd and returns SAS datetime values where the time portion of the value is 000000.</td>
</tr>
<tr>
<td>E8601DTw.d</td>
<td>Informat (p. 544)</td>
<td>Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>E8601Dw.d Informat (p. 546)</td>
<td></td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime extended notation <code>yyyy-mm-ddThh:mm:ss&lt;fff&gt;</code> or <code>yyyy-mm-ddThh:mm:ss&lt;fff&gt;Z</code>.</td>
</tr>
<tr>
<td>E8601Lw.d Informat (p. 548)</td>
<td></td>
<td>Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation <code>hh:mm:ss±hh:mm.&lt;fff&gt;</code> or <code>hh:mm:ss.&lt;fff&gt;Z</code> and converts the values to the local time.</td>
</tr>
<tr>
<td>E8601TMw.d Informat (p. 550)</td>
<td></td>
<td>Reads time values that are specified using the ISO 8601 extended notation <code>hh:mm:ss&lt;fff&gt;</code>.</td>
</tr>
<tr>
<td>E8601TZw.d Informat (p. 552)</td>
<td></td>
<td>Reads time values that are specified using the ISO 8601 extended time notation <code>hh:mm:ss±hh:mm.&lt;fff&gt;</code> or <code>hh:mm:ssZ</code>.</td>
</tr>
<tr>
<td>HHMMSSw. Informat (p. 556)</td>
<td></td>
<td>Reads hours, minutes, and seconds in the form <code>hh:mm:ss</code> or <code>hhmmss</code>.</td>
</tr>
<tr>
<td>JULIANw. Informat (p. 562)</td>
<td></td>
<td>Reads Julian dates in the form <code>yyddd</code> or <code>yyyyyddd</code>.</td>
</tr>
<tr>
<td>MDYAMPw.d Informat (p. 563)</td>
<td></td>
<td>Reads datetime values in the form <code>mm-dd-yyyy&lt;yy&gt;</code> or <code>hh:mm:ss:ss AM/PM</code>, 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.</td>
</tr>
<tr>
<td>MMDDYYw. Informat (p. 565)</td>
<td></td>
<td>Reads date values in the form <code>mmddyy</code> or <code>mmddyyyy</code>.</td>
</tr>
<tr>
<td>MONYYw. Informat (p. 567)</td>
<td></td>
<td>Reads month and year date values in the form <code>mmmyy</code> or <code>mmmyyyy</code>.</td>
</tr>
<tr>
<td>MSECw. Informat (p. 568)</td>
<td></td>
<td>Reads TIME MIC values.</td>
</tr>
<tr>
<td>PDJULGw. Informat (p. 573)</td>
<td></td>
<td>Reads packed Julian date values in the hexadecimal form <code>yyyyyyyyydddf</code> for IBM.</td>
</tr>
<tr>
<td>PDJULlw. Informat (p. 575)</td>
<td></td>
<td>Reads packed Julian dates in the hexadecimal format <code>ccyyyyyyydddf</code> for IBM.</td>
</tr>
<tr>
<td>PDTIMEw. Informat (p. 576)</td>
<td></td>
<td>Reads packed decimal time of SMF and RMF records.</td>
</tr>
<tr>
<td>RMFDURw. Informat (p. 585)</td>
<td></td>
<td>Reads duration intervals of RMF records.</td>
</tr>
<tr>
<td>RMFSTAMPw. Informat (p. 587)</td>
<td></td>
<td>Reads time and date fields of RMF records.</td>
</tr>
<tr>
<td>SHRSTAMPw. Informat (p. 606)</td>
<td></td>
<td>Reads date and time values of SHR records.</td>
</tr>
<tr>
<td>SMFSTAMPw. Informat (p. 607)</td>
<td></td>
<td>Reads time and date values of SMF records.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>STIMERw. Informat (p. 609)</td>
<td>Reads time values and determines whether the values are hours, minutes, or seconds; reads the output of the STIMER system option.</td>
</tr>
<tr>
<td></td>
<td>TIMEw. Informat (p. 610)</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>TODSTAMPw. Informat (p. 612)</td>
<td>Reads an 8-byte time-of-day stamp.</td>
</tr>
<tr>
<td></td>
<td>TUw. Informat (p. 614)</td>
<td>Reads timer units.</td>
</tr>
<tr>
<td></td>
<td>WEEKUw. Informat (p. 618)</td>
<td>Reads a value in the form of a week number within the year and returns a SAS date value by using the U algorithm.</td>
</tr>
<tr>
<td></td>
<td>WEEKVw. Informat (p. 620)</td>
<td>Reads a value in the form of a week number within the year and returns a SAS date value by using the V algorithm.</td>
</tr>
<tr>
<td></td>
<td>WEEKWw. Informat (p. 622)</td>
<td>Reads a value in the form of a week number within the year and returns a SAS date value by using the W algorithm.</td>
</tr>
<tr>
<td></td>
<td>YMDDTTMw.d Informat (p. 624)</td>
<td>Reads datetime values in the form &lt;yy&gt;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.</td>
</tr>
<tr>
<td></td>
<td>YYMMDDw. Informat (p. 626)</td>
<td>Reads date values in the form yyyyymmdd or yyyyymmdd.</td>
</tr>
<tr>
<td></td>
<td>YYMMNw. Informat (p. 628)</td>
<td>Reads date values in the form yyyyymm or yymm.</td>
</tr>
<tr>
<td></td>
<td>YYQw. Informat (p. 630)</td>
<td>Reads quarters of the year in the form yyyyQq or yyyyQq.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>SN8601Bw.d Informat (p. 497)</td>
<td>Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notation.</td>
</tr>
<tr>
<td></td>
<td>SN8601Ew.d Informat (p. 499)</td>
<td>Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.</td>
</tr>
<tr>
<td></td>
<td>B8601Clw.d Informat (p. 517)</td>
<td>Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss&lt;fff&gt;.</td>
</tr>
<tr>
<td></td>
<td>B8601DAw. Informat (p. 518)</td>
<td>Reads date values that are specified using the ISO 8601 base notation yyyyymmdd.</td>
</tr>
<tr>
<td></td>
<td>B8601DJw.d Informat (p. 520)</td>
<td>Reads a Java date and time value that is in the form yyyyymmddhhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td></td>
<td>B8601DNw. Informat (p. 521)</td>
<td>Reads date values that are specified using the ISO 8601 basic notation yyyyymmdd and returns SAS datetime values where the time portion of the value is 000000.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
|          | B8601DTw.d Informat (p. 523) | Reads datetime values that are specified using the ISO 8601 basic notation `yyyymmdTyyyyTmss<ffffff>`.
|          | B8601DZw.d Informat (p. 525) | Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation `yyyymmdTyyyyTmss<ffffff>` or `yyyymmdTyyyyTmss<ffffff>`. |
|          | B8601TMw.d Informat (p. 527) | Reads time values that are specified using the ISO 8601 basic notation `yyyyTmss<ffffff>`.
<p>|          | B8601TZw.d Informat (p. 528) | Reads time values that are specified using the ISO 8601 basic time notation <code>yyyyTmss&lt;ffffff&gt;</code> or <code>yyyyTmss&lt;ffffff&gt;</code>. |
|          | E8601DAw. Informat (p. 542) | Reads date values that are specified using the ISO 8601 extended notation <code>yyyy-mm-dd</code>. |
|          | E8601DNw. Informat (p. 543) | Reads date values that are specified using the ISO 8601 extended notation <code>yyyy-mm-dd</code> and returns SAS datetime values where the time portion of the value is 000000. |
|          | E8601DTw.d Informat (p. 544) | Reads date/time values that are specified using the ISO 8601 extended notation <code>yyyy-mm-ddThh:mm:ss&lt;ffffff&gt;</code>. |
|          | E8601DZw.d Informat (p. 546) | Reads Coordinated Universal Time (UTC) date/time values that are specified using the ISO 8601 datetime extended notation <code>yyyy-mm-ddThh:mm:ss&lt;ffffff&gt;</code> or <code>yyyy-mm-ddThh:mm:ss&lt;ffffff&gt;</code>. |
|          | E8601LZw.d Informat (p. 548) | Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation <code>hh:mm:ss&lt;ffffff&gt;</code> or <code>hh:mm:ss&lt;ffffff&gt;</code> and converts the values to the local time. |
|          | E8601TMw.d Informat (p. 550) | Reads time values that are specified using the ISO 8601 extended notation <code>hh:mm:ss&lt;ffffff&gt;</code>. |
|          | E8601TZw.d Informat (p. 552) | Reads time values that are specified using the ISO 8601 extended time notation <code>hh:mm:ss&lt;ffffff&gt;</code> or <code>hh:mm:ssZ</code>. |
| Numeric  | BINARYw.d Informat (p. 530) | Converts positive binary values to integers. |
|          | BITSw.d Informat (p. 531) | Extracts bits. |
|          | BZw.d Informat (p. 532) | Converts blanks to 0s. |
|          | COMMAw.d Informat (p. 535) | Removes embedded characters. |
|          | COMMAXw.d Informat (p. 536) | 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. |</p>
<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FLOATw.d Informat (p. 554)</td>
<td>Reads a native single-precision, floating-point value and divides it by 10 raised to the $d$th power.</td>
<td></td>
</tr>
<tr>
<td>HEXw. Informat (p. 555)</td>
<td>Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values.</td>
<td></td>
</tr>
<tr>
<td>IBw.d Informat (p. 558)</td>
<td>Reads native integer binary (fixed-point) values, including negative values.</td>
<td></td>
</tr>
<tr>
<td>IBRw.d Informat (p. 559)</td>
<td>Reads integer binary (fixed-point) values in Intel and DEC formats.</td>
<td></td>
</tr>
<tr>
<td>IEEEw.d Informat (p. 561)</td>
<td>Reads an IEEE floating-point value and divides it by 10 raised to the $d$th power.</td>
<td></td>
</tr>
<tr>
<td>NUMXw.d Informat (p. 569)</td>
<td>Reads numeric values with a comma in place of the decimal point.</td>
<td></td>
</tr>
<tr>
<td>OCTALw.d Informat (p. 571)</td>
<td>Converts positive octal values to integers.</td>
<td></td>
</tr>
<tr>
<td>PDw.d Informat (p. 572)</td>
<td>Reads data that is stored in IBM packed decimal format.</td>
<td></td>
</tr>
<tr>
<td>PERCENTw.d Informat (p. 577)</td>
<td>Reads percentages as numeric values.</td>
<td></td>
</tr>
<tr>
<td>PIBw.d Informat (p. 578)</td>
<td>Reads positive integer binary (fixed-point) values.</td>
<td></td>
</tr>
<tr>
<td>PIBRw.d Informat (p. 580)</td>
<td>Reads positive integer binary (fixed-point) values in Intel and DEC formats.</td>
<td></td>
</tr>
<tr>
<td>PKw.d Informat (p. 582)</td>
<td>Reads unsigned packed decimal data.</td>
<td></td>
</tr>
<tr>
<td>RBw.d Informat (p. 584)</td>
<td>Reads numeric data that is stored in real binary (floating-point) notation.</td>
<td></td>
</tr>
<tr>
<td>S370FFw.d Informat (p. 590)</td>
<td>Reads EBCDIC numeric data.</td>
<td></td>
</tr>
<tr>
<td>S370FIBw.d Informat (p. 591)</td>
<td>Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FIBUw.d Informat (p. 592)</td>
<td>Reads unsigned integer binary (fixed-point) values in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPDw.d Informat (p. 594)</td>
<td>Reads packed data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPDUw.d Informat (p. 595)</td>
<td>Reads unsigned packed decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPIBw.d Informat (p. 596)</td>
<td>Reads positive integer binary (fixed-point) values in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FRBw.d Informat (p. 598)</td>
<td>Reads real binary (floating-point) data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDw.d Informat (p. 599)</td>
<td>Reads zoned decimal data in IBM mainframe format.</td>
<td></td>
</tr>
</tbody>
</table>
### Dictionary

**$ASCIIw. Informat**

Converts ASCII character data to native format.

**Categories:**
- CAS
- Character

**Syntax**

$ASCIIw$

**Syntax Description**

*w*  
specifies the width of the input field.

$BASE64Xw. Informat

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

**Categories:** CAS Character

**Alignment:** Left

**Syntax**

$BASE64Xw.

---

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>EBCDIC</td>
<td>ASCII</td>
</tr>
<tr>
<td>abc</td>
<td>a12b3</td>
</tr>
<tr>
<td>ABC</td>
<td>C1C2C3</td>
</tr>
<tr>
<td></td>
<td>414243</td>
</tr>
</tbody>
</table>

* 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.
**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**

```sas
data _null_;  
B64EXMPL="RkNBMDFBNzk5M0JD";  
B64EXMPL2=input(B64EXMPL, $base64x64.);  
put B64EXMPL / B64EXMPL2 ;  
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>B64EXMPL</td>
<td>RkNBMDFBNzk5M0JD</td>
</tr>
<tr>
<td>B64EXMPL2</td>
<td>FCA01A7993BC</td>
</tr>
</tbody>
</table>

**See Also**

- The XMLDOUBLE option of the “LIBNAME Statement for the XMLV2 and XML Engines” in *SAS XMLV2 and XML LIBNAME Engines: User’s Guide*

**Formats:**

- “$BASE64Xw. Format” on page 94

---

**$BINARw. Informat**

Converts binary data to character data.
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, $BINARY_{4}$ 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;

0100110001001101
LM
$CBw$. Informat

Reads standard character data from column-binary files.

**Categories:**
- CAS
- Column Binary

**Syntax**

$CBw$.

**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 $CBw$. informat reads standard character data from column-binary files, where each card column is represented in 2 bytes. The $CBw$. 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**

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------+--------</td>
<td></td>
</tr>
<tr>
<td>0100110001001101</td>
<td>LM</td>
</tr>
</tbody>
</table>

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

- “How to Read Column-Binary Data” in SAS Language Reference: Concepts

Informats:

- “CBw.d Informat” on page 533
- “PUNCH.d Informat” on page 583
- “ROWw.d Informat” on page 588

$CHARw. Informat

Reads character data with blanks.

Categories:
- CAS
- Character

Syntax

$CHARw.

Syntax Description

$w

specifies the width of the input field.

Default

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

Range

1–32767

Details

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

Comparisons

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

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

**Example**

```plaintext
data one;
  name=' X YZ';
  name2=translate(input(name,$char5.),'#', ' ');
  put name ;
  put name2 ;
run;
```

The character # represents a blank space.

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
<td>----------</td>
</tr>
<tr>
<td>X YZ</td>
<td>X YZ</td>
</tr>
<tr>
<td>X YZ</td>
<td>#X#YZ</td>
</tr>
</tbody>
</table>

**$CHARZBw. Informat**

Converts binary 0s to blanks.

**Categories:** CAS

**Syntax**

$CHARZBw.

**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.
Details

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

Comparisons

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

Example

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

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

* The data lines are hexadecimal representations of codes for characters. Each two hexadecimal characters correspond to 1 byte of binary data, and each byte corresponds to one character.

** The character # represents a blank space.

$EBCDICw. Informat

Converts EBCDIC character data to native format.

Syntax

$EBCDICw.

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

Example

input @1 name $ebcdic3.

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ASCII</td>
</tr>
<tr>
<td>-----</td>
<td></td>
</tr>
<tr>
<td>qrs</td>
<td>717273</td>
</tr>
<tr>
<td>QRS</td>
<td>515253</td>
</tr>
<tr>
<td>+;/&gt;</td>
<td>2B3B3E</td>
</tr>
</tbody>
</table>

* 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: Character

See: "$HEX Informat: UNIX" in SAS Companion for UNIX Environments
"$HEXw. Informat: Windows" in SAS Companion for Windows

Syntax

$HEXw.
**Syntax Description**

$w$

specifies the number of digits of hexadecimal data.

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

Default 2

Range 1–32767

**Details**

The $\text{HEX}_w$. informat converts every two digits of hexadecimal data to 1 byte of character data. Use $\text{HEX}_w$. to encode hexadecimal values into a character variable when your input method is limited to printable characters.

**Comparisons**

The HEXw. informat reads two digits of hexadecimal data at a time and converts them to 1 byte of numeric data.

**Example**

```plaintext
data test;
  name='6C6C';
  name2=input(name,$hex4.);
run;
```

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
<th>ASCII</th>
<th>EBCDIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>6C6C</td>
<td>11</td>
<td>%%</td>
<td></td>
</tr>
</tbody>
</table>

**$N8601Bw.d$ 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

Date and Time

ISO 8601

**Alignment:** Left

**Restriction:** UTC time zone offset values are not supported.

**Supports:** ISO 8601 Element 5.4.4, complete representation
Syntax

SN8601B\(w,d\)

**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 SN8601B informat reads ISO 8601 duration, interval, and datetime values as character data for these basic notations:

<table>
<thead>
<tr>
<th>Time Component</th>
<th>ISO 8601 Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>yyyy-mm-ddThh:mm:ss.fff</td>
<td>P2012-09-15T15:53:00</td>
</tr>
<tr>
<td></td>
<td>yyyy-mm-ddThh:mm:ss.fff</td>
<td>P00020304T050607</td>
</tr>
<tr>
<td></td>
<td>yyyy-mm-ddThh:mm:ss.fff</td>
<td>P2y10m14dT20h13m45.222s</td>
</tr>
<tr>
<td>Interval</td>
<td>yyyy-mm-ddT hh:mm:ss.fff/ yyyy-mm-ddT hh:mm:ss.fff</td>
<td>2012-09-15T15:53:00/2014-11-13T00:00:00</td>
</tr>
<tr>
<td></td>
<td>yyyy-mm-ddT hh:mm:ss.fff</td>
<td>20120915T155300/20141115T120000</td>
</tr>
<tr>
<td></td>
<td>yyyy-mm-ddT hh:mm:ss.fff</td>
<td>P2y10M14dT20h13m45s/2012-09-15T15:53:00</td>
</tr>
<tr>
<td>Datetime</td>
<td>yyyy-mm-ddT hh:mm:ss.fff</td>
<td>2012-09-15T15:53:00</td>
</tr>
<tr>
<td></td>
<td>yyyy-mm-ddT hh:mm:ss.fff</td>
<td>20120915T155300</td>
</tr>
</tbody>
</table>
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

data one;
  date='2018-09-15T15:53:00';
  date2=input(date,$n8601b20.);

  put date /
      date2 ;
run;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date='2018-09-15T15:53:00';</td>
<td>2018-09-15T15:53:00</td>
</tr>
<tr>
<td>date2=input(date,$n8601b20.);</td>
<td>2018915155300FFD</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

$N8601Ew.d Informat

Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.

Categories: CAS
            Date and Time
            ISO 8601

Alignment: Left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.4, complete representation
Syntax

$N8601E_w.d$

Syntax Description

\( w \)

specifies the width of the input field.

- **Default**: 50
- **Range**: 1–200
- **Requirement**: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

\( d \)

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

- **Default**: 0
- **Range**: 0–3

Details

The $N8601E$ informat reads ISO 8601 duration, interval, and datetime values that can be specified in these extended notations:

<table>
<thead>
<tr>
<th>Time Component</th>
<th>ISO 8601 Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>Pyyyy-mm-ddThh:mm:ss.fff</td>
<td>P2012-09-15T15:53:00</td>
</tr>
<tr>
<td></td>
<td>PnW</td>
<td>P6w</td>
</tr>
<tr>
<td>Interval</td>
<td>yyyy-mm-ddThh:mm:ss.fff</td>
<td>2012-09-15T15:53:00/2014-1</td>
</tr>
<tr>
<td></td>
<td>yyyy-mm-ddThh:mm:ss.fff</td>
<td>1-13T00:00:00</td>
</tr>
<tr>
<td>Datetime</td>
<td>yyyy-mm-ddThh:mm:ss.fff</td>
<td>2012-09-15T15:53:00</td>
</tr>
</tbody>
</table>

\( n \)

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**

```plaintext
data one;
  date='2018-09-15T15:53:00';
  date2=input(date, $n8601e20.);

  put date / date2 ;
run;
```
See Also
“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

$OCTALw. Informat

Converts octal data to character data.

Categories:
CAS
Character

Syntax

$OCTALw.

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

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

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

Comparisons

The OCTALw. informat reads octal data and converts it to the numeric equivalents.
Example

    input @1 name $octal9.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>114</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

$PHEXw. Informat

Converts packed hexadecimal data to character data.

Categories: CAS
Character

Syntax

$PHEXw.

Syntax Description

w

specifies the number of bytes in the input.

When you use $PHEXw. to read packed hexadecimal data, the length of the variable is the number of bytes that are required to store the resulting character value, not w. In general, a character variable whose length is implicitly defined with $PHEXw. has a length of 2w−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 $PHEXw. 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 PDw.d. informat reads packed decimal data and converts it to numeric data.

Example

    input @1 devaddr $phex2.;
The data line represents 2 bytes of actual binary data. Each half-byte corresponds to a single hexadecimal character. The equivalent hexadecimal representation for the data line is 1E0F.

### $QUOTEw. Informat

Removes matching quotation marks from character data.

**Categories:** CAS
Character

**Syntax**

$QUOTEw.

**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**

```sas
data one;
  name="'SAS'";
  name2=input(name,$quote7.);

  put name= / name2= ;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>'SAS'</td>
<td>SAS</td>
</tr>
</tbody>
</table>

### $UPCASEw. Informat

Converts character data to uppercase.

**Categories:** CAS
Character
Syntax
$UPCASEw.

Syntax Description

w specifies the width of the input field.

Default  8 if the length of the variable is undefined. Otherwise, the 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;

Data Line | Result
---------|------
 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;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1548611f-b8cb-6a47-a721-a6fd284e74f2</td>
<td>1548611FB8CB6A47A721A6FD284E74F2</td>
</tr>
</tbody>
</table>

$VARYINGw. Informat

Reads character data of varying length.

Categories:
- CAS
- Character

Syntax

$VARYINGw. 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 \texttt{length-variable} immediately after \texttt{VARYING w.} in an INPUT statement.

Tips: If the value of \texttt{length-variable} is negative or missing, SAS reads no data from the corresponding record.

If the value of \texttt{length-variable} is 0, the value of the variable is a blank character. A value of 0 for \texttt{length-variable} enables you to read zero-length records and fields.

If a variable has been read using an informat other than the \texttt{VARYING.} informat, and then the same data is read into the same variable that uses the \texttt{VARYING.} informat where \texttt{length-variable} is 0, then the previous value is overwritten with a blank value.

If \texttt{length-variable} is greater than 0 but less than \texttt{w}, SAS reads the number of columns that are specified by \texttt{length-variable}. Then, SAS pads the value with trailing blanks up to the maximum width that is assigned to the variable.

If \texttt{length-variable} is greater than or equal to \texttt{w}, SAS reads \texttt{w} columns.

Details

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

Examples

\textbf{Example 1: Obtaining a Current Record Length Directly}

```
data one;
   input fwidth 1. name $varying9. fwidth;
   put name;
   datalines;
5shark
3sunfish
8bluefish
;
run;
```

<table>
<thead>
<tr>
<th>Data Record from Text File</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5shark</td>
<td>shark</td>
</tr>
<tr>
<td>3sunfish</td>
<td>sun</td>
</tr>
<tr>
<td>8bluefish</td>
<td>bluefish</td>
</tr>
</tbody>
</table>

\* Notice the result of reading the second data line.
Example 2: Obtaining a Current Record Length Indirectly

Use the LENGTH= option in the INFILE statement to obtain a record length indirectly. The input data lines and results follow the explanation of the SAS statements.

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

The LENGTH= option in the INFILE statement assigns the internally stored record length to 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.

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

$w.$ Informat

Reads standard character data.

- **Categories:** CAS
- **Character**
- **Alias:** $Fw.$

**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 $CHARw$. informat. However, $CHARw$. does not trim leading blanks nor does it convert a single period in an input field to a blank, whereas $w$. does both.

Examples

Example 1

```
data one;
  name=' XYZ '; 
  name2=input(name,$5.);

  put name / 
    name2 ; 
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>XYZ</td>
</tr>
</tbody>
</table>

Example 2

This example represents blank spaces with #.

```
data one;
  name=' XYZ '; 
  name2=input(name,$5.);

  name=translate(name,'#',' '); 
  name2=translate(name2,'#',' '); 

  put name / 
    name2 ; 
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>#XYZ#</td>
<td>XYZ##</td>
</tr>
</tbody>
</table>

ANYDTDTEw. Informat

Reads and extracts the date value from various date, time, and datetime forms.
Syntax

\textsc{anydtdte}_w. 

\textbf{Syntax Description}

\(w\) specifies the width of the input field.

\begin{itemize}
  \item \textbf{Default} 9
  \item \textbf{Range} 5–60
\end{itemize}

\textbf{Details}

The \textsc{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.

<table>
<thead>
<tr>
<th>Informat or Form of Input</th>
<th>Example Data</th>
<th>Informat or Form of Input</th>
<th>Example Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>01JAN18</td>
<td>MONYY</td>
<td>JAN18</td>
</tr>
<tr>
<td></td>
<td>01JAN2018</td>
<td></td>
<td>JAN2018</td>
</tr>
<tr>
<td>DATETIME</td>
<td>01JAN18 14:30:08</td>
<td>TIME</td>
<td>14:30</td>
</tr>
<tr>
<td></td>
<td>01JAN2018 14:30:08.5</td>
<td></td>
<td>14:30:08.05</td>
</tr>
<tr>
<td>DDMMYY</td>
<td>010118</td>
<td>YMDDDTTM</td>
<td>18-01-01 11:23</td>
</tr>
<tr>
<td></td>
<td>01012018</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULIAN</td>
<td>18001</td>
<td>YYMDD</td>
<td>180101</td>
</tr>
<tr>
<td></td>
<td>2018001</td>
<td></td>
<td>20180101</td>
</tr>
<tr>
<td>MDYAMPM</td>
<td>01-01-18 3:53 pm</td>
<td>YYQ</td>
<td>18Q1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2018Q1</td>
</tr>
<tr>
<td>MMDDYY</td>
<td>010118</td>
<td>YY&lt;YY&gt;xMM (^*)</td>
<td>18/01</td>
</tr>
<tr>
<td></td>
<td>01012018</td>
<td></td>
<td>2018-01</td>
</tr>
<tr>
<td>MMxYY&lt;YY&gt; (^*)</td>
<td>01/18</td>
<td>month-day-year</td>
<td>January 1, 2018</td>
</tr>
<tr>
<td></td>
<td>01-2018</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\(^*\) \(x\) 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 ANYTDDE informat extracts the date part from the derived value. The ANYTDTM informat extracts the datetime part. The ANYTTME informat extracts the time part.

**Example**

```sas
data test;
  dateinfo='01JAN2018 14:30:08.5';
  sasdate=input(dateinfo,anytdtde21.);
  put sasdate ;
  put sasdate date9.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate ;</td>
<td>21185</td>
</tr>
<tr>
<td>put sasdate date9.;</td>
<td>01JAN2018</td>
</tr>
</tbody>
</table>

**See Also**

**Informats:**
- “ANYTDDEw. Informat” on page 511
- “ANYDDEw. Informat” on page 515
- “DATEw. Informat” on page 537
- “DATETIMEw. Informat” on page 538
- “DDMMYYw. Informat” on page 540
- “JULIANw. Informat” on page 562
- “MDYAMPW.d Informat” on page 563
- “MMDDYYw. Informat” on page 565
- “MONYYw. Informat” on page 567
- “TIMEw. Informat” on page 610
- “YMDDTTMw.d Informat” on page 624
- “YYMMDDw. Informat” on page 626
- “YYQw. Informat” on page 630
ANYDTDTMw. Informat

Reads and extracts datetime values from various date, time, and datetime forms.

**Categories:**
CAS
Date and Time

**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 ANYDTDTMw. 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**

ANYDTDTMw.

**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:

<table>
<thead>
<tr>
<th>Informat or Form of Input</th>
<th>Example Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>01JAN18</td>
</tr>
<tr>
<td></td>
<td>01JAN2018</td>
</tr>
<tr>
<td>DATETIME</td>
<td>01JAN18 14:30:08</td>
</tr>
<tr>
<td></td>
<td>01JAN2018 14:30:08.5</td>
</tr>
<tr>
<td>DDMM&lt;YY&gt;YY</td>
<td>010118</td>
</tr>
<tr>
<td></td>
<td>01012018</td>
</tr>
<tr>
<td>JULIAN</td>
<td>18001</td>
</tr>
<tr>
<td></td>
<td>2018001</td>
</tr>
<tr>
<td>MMDD&lt;YY&gt;YY</td>
<td>010118</td>
</tr>
<tr>
<td></td>
<td>01012018</td>
</tr>
<tr>
<td>MM&lt;YY&gt;YY</td>
<td>01/18</td>
</tr>
<tr>
<td></td>
<td>01-2018</td>
</tr>
<tr>
<td>Informat or Form of Input</td>
<td>Example Data</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>
| MDYAMPM ""              | 01/01/18 02:30:08 AM  
                          | 01/01/2018 02:30:08 AM |
| MONYY                   | JAN18       
                          | JAN2018       |
| TIME                    | 14.30       
                          | 14:30:08.05   |
| <YY>YYMDD ""           | 180101      
                          | 20180101      |
| <YY>YYQ ""             | 18Q1        
                          | 2018Q1        |
| <YY>YY:MM ""           | 18/01       
                          | 2018/01       |

* x is a special character that separates the month from the year.

** <YY> indicates the century is optional.

*** 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 YYMMD 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:

<table>
<thead>
<tr>
<th>Use of Colons and Periods</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>A single colon in the value h:m indicates hours and minutes.</td>
<td>14:30</td>
</tr>
<tr>
<td>Two colons in the value h:m:s indicate hours, minutes, and seconds.</td>
<td>14:30:08</td>
</tr>
</tbody>
</table>
Use of Colons and Periods

<table>
<thead>
<tr>
<th>Example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2:39.66</td>
<td>A single period in the value m:s.ff, where ff is a fraction of a second, indicates that the number preceding the period is the number of seconds.</td>
</tr>
<tr>
<td>12.25.2018</td>
<td>Multiple periods in the value indicate that the period is a delimiter for dates and the value is not a time value.</td>
</tr>
</tbody>
</table>

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

```sas
data test;
  dateinfo='01JAN2018 14:30:08.5';
  sasdate=input(dateinfo,anydtfmt21.);
  put sasdate ;
  put sasdate datetime20.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Formatted with DATETIMEw.d Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate datetime20.;</td>
<td>1830436208.5</td>
<td>01JAN2018:14:30:09</td>
</tr>
</tbody>
</table>

See Also

Informats:

- “ANYDTDTEw. Informat” on page 509
- “ANYDTTMEw. Informat” on page 515
- “DATEw. Informat” on page 537
- “DATETIMEw. Informat” on page 538
- “DDMMYYw. Informat” on page 540
- “JULIANw. Informat” on page 562
- “MMDDYYw. Informat” on page 565
- “MONYYw. Informat” on page 567
- “TIMEw. Informat” on page 610
- “YYMMDDw. Informat” on page 626
- “YYQw. Informat” on page 630
ANYDTTMEw. Informat

Reads and extracts time values from various date, time, and datetime forms.

**Categories:**
- CAS
- Date and Time

**Syntax**

```
ANYDTTMEw.
```

**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:

<table>
<thead>
<tr>
<th>Informat or Form of Input</th>
<th>Example Data</th>
<th>Informat or Form of Input</th>
<th>Example Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>01JAN18</td>
<td>MONYY</td>
<td>JAN18</td>
</tr>
<tr>
<td></td>
<td>01JAN2018</td>
<td></td>
<td>JAN2018</td>
</tr>
<tr>
<td>DATETIME</td>
<td>01JAN18 14:30:08</td>
<td>YYMDD</td>
<td>180101</td>
</tr>
<tr>
<td></td>
<td>01JAN2018 14:30:08.5</td>
<td></td>
<td>20180101</td>
</tr>
<tr>
<td>DDMMYY</td>
<td>010118</td>
<td>YYQ</td>
<td>18Q1</td>
</tr>
<tr>
<td></td>
<td>01012018</td>
<td></td>
<td>2018Q1</td>
</tr>
<tr>
<td>JULIAN</td>
<td>18001</td>
<td>YYQ</td>
<td>18Q1</td>
</tr>
<tr>
<td></td>
<td>2018001</td>
<td></td>
<td>2018Q1</td>
</tr>
<tr>
<td>MMDDYY</td>
<td>010118</td>
<td><code>month-day-year</code></td>
<td>January 1, 2018</td>
</tr>
<tr>
<td></td>
<td>01012018</td>
<td></td>
<td>2018-01</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Use of Colons and Periods</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>A single colon in the value h:m indicates hours and minutes.</td>
<td>14:30</td>
</tr>
<tr>
<td>Two colons in the value h:m:s indicate hours, minutes, and seconds.</td>
<td>14:30:08</td>
</tr>
<tr>
<td>A single period in the value m:s.ff, where ff is a fraction of a second, indicates that the number preceding the period is the number of seconds.</td>
<td>2:39.66</td>
</tr>
<tr>
<td>Multiple periods in the value indicate that the period is a delimiter for dates and the value is not a time value.</td>
<td>12.25.2018</td>
</tr>
</tbody>
</table>

**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**

```plaintext
data test;
  dateinfo='01JAN2018 14:30:08.5';
  sastime=input(dateinfo, anydttme21.);
  put sastime ;
  put sastime time8.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Informat</th>
<th>Result</th>
<th>Formatted with the TIMEw.d Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sastime time8.;</td>
<td>DATETIME</td>
<td>52208.5</td>
<td>14:30:09</td>
</tr>
</tbody>
</table>

**See Also**

- “ANYDTDTEw. Informat” on page 509
- “ANYDTDTMw. Informat” on page 512
- “DATEw. Informat” on page 537
- “DATETIMEw. Informat” on page 538
- “DDMMYYw. Informat” on page 540
- “JULIANw. Informat” on page 562
B8601CIw.d Informat

Reads an IBM date and time value that includes a century marker, in the form cyymmdhhmmss<fff>.

Categories: Date and Time
ISO 8601

Alignment: Left

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

Syntax

B8601CIw.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 cyymmdhhmmss<fff>.

c

is a single digit that represents a century.

0 indicates the years 1900–1999.
1 indicates the years 2000–2099.
2 indicates the years 2100–2199.

n indicates the years 00–99 in a century that is determined by performing a calculation on a year greater than 2199. To determine the century marker, subtract 1900 from the year and divide the result by 100. Discard the remainder. The remaining integer is the century marker. For example, to
determine the century marker for the year 2382, perform this calculation:
\[(2382-1900)/100=4.82\]. Discard .82. The century marker is 4.

\(yy\)
is a two-digit year between 00 and 99.

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

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

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

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

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

\(fff\)
are optional fractional seconds, with a precision of up to three digits, where each
digit is between 0 and 9.

**Example**

```plaintext
data one;
    date='118100112153025';
    date2=input(date,B8601CI.);

    put date /
          date2 /
          date2 datetime25.2;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>118100112153025</td>
</tr>
<tr>
<td>date2</td>
<td>1854015330.3</td>
</tr>
<tr>
<td>date2=input(date,B8601CI.);</td>
<td>01OCT2018:12:15:30.25</td>
</tr>
</tbody>
</table>

**B8601DAw. Informat**

Reads date values that are specified using the ISO 8601 base notation \(yyyyymmdd\).

**Categories:** Date and Time

ISO 8601

**Alignment:** Left

**Alias:** ND8601DAw
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

B8601DAw.

Syntax Description

w
specifies the width of the input field.

Default 10

Requirement The width of the output field must be 10.

Details

The B8601DA informat reads date values that are specified using the ISO 8601 basic date notation yyyyymmdd.

yyyy
is a four-digit year.

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

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

If the month or day values are omitted, SAS uses a value of 1 for the month or day. If the hour, minute, or second values are omitted, SAS uses a value of 0 for the hour, minute, or second.

Example

```
data _null_;  
  input d b8601da.;  
  put d;  
  datalines;  
20180915  
2018  
;  
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>20180915</td>
<td>21442</td>
</tr>
<tr>
<td>2018</td>
<td>21885</td>
</tr>
</tbody>
</table>
See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

**B8601DJw.d Informat**

Reads a Java date and time value that is in the form `yyyyymmddhhmmss<ffffff>`.

**Categories:** Date and Time  
ISO 8601

**Alignment:** Left

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

**Syntax**

`B8601DJw.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 B8601DJw.d informat reads a date and time value that is specified using the Java date and time notation `yyyyymmddhhmmss<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.
are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Comparisons

The B8601DJw.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

```sas
data one;
  date='2018090107300033';
  date2=input(date,B8601DJ.);
  put date  /
         date2 /
         date2 datetime25.2;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>2018090107300033</td>
</tr>
<tr>
<td>date2</td>
<td>1851406200.3</td>
</tr>
<tr>
<td>date2 datetime25.2;</td>
<td>01SEP2018:07:30:00.33</td>
</tr>
</tbody>
</table>

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 and Time
  - ISO 8601
- **Alignment:** Left
- **Alias:** ND8601DNw
- **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<sup>w</sup>

Syntax Description

<sup>w</sup> specifies the width of the input field.

Default

<table>
<thead>
<tr>
<th>Requirement</th>
<th>10</th>
</tr>
</thead>
</table>

Details

The B8601DN informat reads date values that are specified using the ISO 8601 basic date notation `yyyyymmdd` 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';
date2=input(date,B8601DN.);

put date /
    date2 /
    date2 datetime20.;
run;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>20180915</td>
</tr>
<tr>
<td>date2</td>
<td>1852588800</td>
</tr>
<tr>
<td>date2=input(date,B8601DN.);</td>
<td>15SEP2018:00:00:00</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467
B8601DT \textit{w.d} Informat

Reads datetime values that are specified using the ISO 8601 basic notation \texttt{yyyymmddThhmmss<ffffff>}.  

\textbf{Categories:} Date and Time  
ISO 8601  

\textbf{Alignment:} Left  

\textbf{Aliases:} B8601L\textit{w.d}  
ND8601DT\textit{w.d}  

\textbf{Restrictions:} This informat is not supported in a DATA step that runs in CAS.  UTC time zone offset values are not supported.

\section*{Syntax}

\texttt{B8601DT \textit{w.d}}

\section*{Syntax Description}

\textbf{\textit{w}}

specifies the width of the input field.

Default 19  
Range 14–26

\textbf{\textit{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

\section*{Details}

The B8601DT informat reads datetime values that are specified in the ISO 8601 basic datetime notation \texttt{yyyymmddThhmmss<ffffff>}.  

\texttt{yyyy}

is a four-digit year.  

\texttt{mm}

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

\texttt{dd}

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

\texttt{hh}

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

\texttt{mm}

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

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;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>20180915T155300</td>
<td>1852645980</td>
</tr>
<tr>
<td>2018</td>
<td>1830184000</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

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 yyyymmddThhmmss +|-hhmm

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

See: The B8601DXw. informat uses the B8601DZw.d informat to read data. See “B8601DZw.d Informat” on page 525.

Syntax

B8601DXw.
**B8601DZw.d Informat**

Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation `yyyymmddThhmmss[±]hhmm` or `yyyymmddThhmmss<ffffff>Z`.

**Categories:** Date and Time  
ISO 8601

**Alignment:** Left

**Aliases:**  
B8601DXw.d  
ND8601DZw.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**

**B8601DZw.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:

- `yyyymmddThhmmss[±]hhmm`
- `yyyymmddThhmmss<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.

fffffff 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;
  datalines;
  20180915T155300+0500 ;
run;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>20180915T155300+0500</td>
<td>1852627980</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

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 yyyymmddThhmmss+|–hhmm.

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

See: The B8601LXw.d informat uses the B8601DTw.d informat to read data. See “B8601DTw.d Informat” on page 523.
B8601TMw.d Informat

Reads time values that are specified using the ISO 8061 basic notation $hh:mm:ss<ffffff>$.

**Categories:** Date and Time  
ISO 8601  

**Alignment:** Left  

**Alias:** ND8601TMw.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**

B8601TMw.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 8061 basic 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.
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;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>155300</td>
<td>57180</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

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 \texttt{hhmmss}+|–hhmm.

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

See: The B8601TXw. informat uses the B8601TZw.d informat to read data. See “B8601TZw.d Informat” on page 528.

Syntax

B8601TXw.

B8601TZw.d Informat

Reads time values that are specified using the ISO 8601 basic time notation \texttt{hhmmss}<fff>+|–hhmm or \texttt{hhmmss}<ffffff>Z.

Categories: Date and Time
ISO 8601

Alignment: Left

Aliases: B8601TXw.d
ND8601TZw.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:

- \( hhmmss<ffffff>\pm hhmm \)
- \( hhmmss<ffffff>Z \)

- **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.

- \( \pm hh:mm \)
  - is an hour and minute signed offset from zero meridian time. Note that the offset must be \( \pm hhmm \) (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 \( \pm hh \) 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

```sas
data _null_;  
  input d b8601tz.;  
  put d;  
  datalines;  
  202401-0500  
  202401Z  
  202401+0000  
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>202401-0500</td>
<td>5041</td>
</tr>
<tr>
<td>202401Z</td>
<td>73441</td>
</tr>
<tr>
<td>202401+0000</td>
<td>73441</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

**BINARYw.d Informat**

Converts positive binary values to integers.

**Categories:** CAS

**Syntax**

`BINARYw.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.
BINARYw.d ignores leading and trailing blanks.
BINARYw.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;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.5</td>
</tr>
</tbody>
</table>

BITSw.d Informat

Extracts bits.

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

BITSw.d

Syntax Description

w
specifies the number of bits to read.
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**

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>*-------1----+</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>B</td>
</tr>
</tbody>
</table>

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

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

---

**BZ\(_w.d\) Informat**

Converts blanks to 0s.

**Syntax**

\[ BZ_{w.d} \]

**Syntax Description**

\(w\)

specifies the width of the input field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1–32</td>
</tr>
</tbody>
</table>
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 BZw.d informat reads numeric values, converts any trailing or embedded blanks to 0s, and ignores leading blanks.

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

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

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

Comparisons

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

Example

```sas
data _null_;  
  input @1 x bz4.;  
  put x;  
  datalines;  
  34  
  -2  
  -2 1  
run;
```

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

CBw.d Informat

Reads standard numeric values from column-binary files.
Syntax

\[ \text{CB}_w.d \]

**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 \( \text{CB}_w.d \) 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 \( \text{CB}_w.d \) 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.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0009</td>
<td>9</td>
</tr>
</tbody>
</table>

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

**See Also**


**Informs**

- “$CBw. Informat” on page 492
COMMA\textit{w}.\textit{d} Informat

Removes embedded characters.

Categories: CAS
          Numeric

Alias: DOLLAR\textit{w}.\textit{d}

Syntax

\texttt{COMMA}\textit{w}.\textit{d}

\textbf{Syntax Description}

\textit{w}

specifies the width of the input field.

- Default: 1
- Range: 1–32

\textit{d}

specifies the power of 10 by which to divide the value. If the data contains decimal points, the \textit{d} value is ignored. This argument is optional.

- Range: 0–31

\textbf{Details}

The COMMA\textit{w}.\textit{d} informat reads numeric values and removes embedded commas, blanks, dollar signs, percent signs, hyphens, and close parentheses from the input data. The COMMA\textit{w}.\textit{d} informat converts an open parenthesis at the beginning of a field to a minus sign.

\textbf{Comparisons}

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

\textbf{Example}

\begin{verbatim}
data test;
  value='$1,000,000';
  value2=input(value,comma10.);
  put value2 ;
run;
\end{verbatim}
**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:** DOLLARXw.d

**Syntax**

COMMAXw.d

**Syntax Description**

w  
spreads 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 COMMAXw.d informat reads numeric values and removes embedded periods, blanks, dollar signs, percent signs, hyphens, and closing parentheses from the input data. The COMMAXw.d informat converts an open parenthesis at the beginning of a field to a minus sign.

**Comparisons**

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

**Example**

```plaintext
data one;
```
```sas
input @1 x commax10.;
put x;
datalines;
$1.000.000
1.234,56
(500)
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1.000.000</td>
<td>1000000</td>
</tr>
<tr>
<td>1.234,56</td>
<td>1234.56</td>
</tr>
<tr>
<td>(500)</td>
<td>-500</td>
</tr>
</tbody>
</table>

**DATEw. Informat**

Reads date values in the form `ddmmmyy` or `ddmmmyyyy`.

**Categories:**
- CAS
- Date and Time

**Syntax**

`DATEw.`

**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**

```sas
data test;
  date='04MAY2018';
  sasdate=input(date,date9.);
  put sasdate ;
  put sasdate date9.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate ;</td>
<td>21308</td>
</tr>
<tr>
<td>put sasdate date9;</td>
<td>04MAY2018</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “DATEw. Format” on page 150

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

**System Options:**
- “YEARCUTOFF= System Option” in *SAS System Options: Reference*

---

**DATETIMEw. Informat**

Reads datetime values in the form `ddmmyy hh:mm:ss.ss` or `ddmmmyyyyy hh:mm:ss.ss`.

**Categories:**
- CAS
- Date and Time

**Syntax**

`DATETIMEw.`

**Syntax Description**

`w`

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

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

*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 *

Example
```sas
   data test;
      date='04MAY2018:13:33:18';
      sasdate=input(date,datetime.);
      put sasdate ;
      put sasdate datetime20. ;
   run;
```
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 and Time

**Syntax**

```
DDMMYYw.
```

**Syntax Description**

`w`

specifies the width of the input field.
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

```sas
data test;
  date='04/05/2018';
  sasdate=input(date,ddmmyy10.);
  put sasdate ;
  put sasdate date9.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate ;</td>
<td>21308</td>
</tr>
<tr>
<td>put sasdate date9.;</td>
<td>04MAY2018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 150
- “DDMMYYw. Format” on page 156
- “MMDDYYw. Format” on page 207
- “YYMMDDw. Format” on page 430

Functions:
E8601DAw. Informat

Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.

Categories: CAS
Date and Time
ISO 8601

Alignment: Left

Alias: IS8601DAw

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

E8601DAw.

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;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-09-15</td>
<td>21442</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

**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
- Date and Time
- ISO 8601

**Alignment:** Left

**Alias:** IS8601DNw

**Restriction:** UTC time zone offset values are not supported.

**Supports:** ISO 8601 Element 5.2.1.1, complete representation

**Syntax**

\[ \text{E8601DNw}. \]

**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 \texttt{yyyy-mm-dd} and returns the date in a SAS datetime value.

\texttt{yyyy} is a four-digit year.

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

\texttt{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;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------</td>
<td>-----------</td>
</tr>
<tr>
<td>2018-09-15</td>
<td>1852588800</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

E8601DTw.d Informat

Reads datetime values that are specified using the ISO 8601 extended notation \texttt{yyyy-mm-ddThh:mm:ss.<ffffff>}.  

\textbf{Categories:} CAS  
Date and Time  
ISO 8601  

\textbf{Alignment:} Left  

\textbf{Aliases:} E8601LXw.d  
IS8601DTw.d  

\textbf{Restriction:} UTC time zone offset values are not supported.  

\textbf{Supports:} ISO 8601 Element 5.4.1, complete representation
Syntax

E8601DTw.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-ddThh: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;
<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-09-15T15:53:00</td>
<td>1852645980</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

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-ddThh:mm:ss±–hh:mm.

Category: CAS

See: The E8601DXw. informat uses the E8601DZw.d informat to read data. See “E8601DZw.d Informat” on page 546.

Syntax

E8601DXw.

E8601DZw.d Informat

Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime extended notation yyyy-mm-ddThh:mm:ss±–hh:mm.<fff> or yyyy-mm-ddThh:mm:ss.<fff>±Z.

Categories: CAS

Date and Time

ISO 8601

Alignment: Left

Aliases: E8601DXw.d

IS8601DZw.d

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

E8601DZw.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:

- \[ yyyy-mm-ddThh:mm:ss.<ffffff>\pm hh:mm \]
- \[ yyyy-mm-ddThh:mm:ss.<ffffff>Z \]

\[ yyyy \]
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.

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

\[ \pm hh:mm \]
is an hour and minute signed offset from zero meridian time. Note that the offset
must be \[ \pm hh:mm \] (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;
  datalines;

2018-09-15T15:53:00Z
; run;

data one;
  input edz e8601dz28.2;
  put edz;
  datalines;
  2018-09-15T15:53:00+03:00
  ; run;

<table>
<thead>
<tr>
<th>Input Statement</th>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>input edz e8601dz.;</td>
<td>2018-09-15T15:53:00Z</td>
<td>1852645980</td>
</tr>
<tr>
<td>input edz e8601dz28.2;</td>
<td>2018-09-15T15:53:00+03:00</td>
<td>1852635180</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

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-ddThh:mm:ss±|–hh:mm**.

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

See: The E8601LXw. informat uses the E8601DTw.d informat to read data. See “E8601DTw.d Informat” on page 544.

Syntax

E8601LXw.

E8601LZw.d Informat

Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation **hh:mm:ss±|–hh:mm.<ffffff>** or **hh:mm:ss.<ffffff>Z** and converts the values to the local time.

Categories: Date and Time
ISO 8601

Alignment: Left
Alias: IS8601LZw.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

**E8601LZw.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.<ffffffff>+-|–hh:mm`
- `hh:mm:ss.<ffffffff>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.
- `ffffffff` 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

```sas
data one;
  input edz e8601lz.;
  put edz;
datalines;
  09:13:21+02:00
  23:43:44Z
; run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:13:21+02:00</td>
<td>11601</td>
</tr>
<tr>
<td>23:43:44Z</td>
<td>85424</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

E8601TMw.d Informat

Reads time values that are specified using the ISO 8601 extended notation \texttt{hh:mm:ss.<ffffff>}.  

<table>
<thead>
<tr>
<th>Categories</th>
<th>CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date and Time</td>
<td></td>
</tr>
<tr>
<td>ISO 8601</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alignment</th>
<th>Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>IS8601TMw.d</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Restriction</th>
<th>UTC time zone offset values are not supported.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>ISO 8601 Elements 5.3.1.1 and 5.3.1.3, complete representation and representation of decimal fractions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax</th>
<th>E8601TMw.d</th>
</tr>
</thead>
</table>
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

```plaintext
data one;
  input edm e8601tm.;
  put edm;
  datalines;
  15:53:00
; run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:53:00</td>
<td>57180</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467
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 E8601TXw. informat uses the E8601TZw.d informat to read data. See “E8601DZw.d Informat” on page 546.

**Syntax**

E8601TXw.

---

E8601TZw.d Informat

Reads time values that are specified using the ISO 8601 extended time notation `hh:mm:ss+–hh:mm.<fffffff>` or `hh:mm:ssZ`.

**Categories:** CAS
Date and Time
ISO 8601

**Alignment:** Left

**Aliases:** E8601TXw.d
IS8601TZw.d

**Supports:** ISO 8601 Element 5.3.1.1, complete representation

**Syntax**

E8601TZw.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:

- \( hh:mm:ss+\text{|–}hh:mm.<ffffff> \)
- \( hh:mm:ssZ \)

\( hh \)
- 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.

\( +\text{|–}hh:mm \)
- is an hour and minute signed offset from zero meridian. Note that the offset must be \( +\text{|–}hh:mm \) (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 \( +\text{|–}hh \) 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;
  datalines;
  23:43:44–05:00
  23:43:44Z
  ;
  run;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>23:43:44–05:00</td>
<td>17024</td>
</tr>
<tr>
<td>23:43:44Z</td>
<td>85424</td>
</tr>
</tbody>
</table>
See Also

“Reading Dates and Times by Using the ISO 8601 Basic and Extended Notations” on page 467

**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**

`FLOATw.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 `FLOATw.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:

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

**Example**

```
input x float4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3F800000</td>
<td>1</td>
</tr>
</tbody>
</table>

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

---

**HEXw. 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. Informat: Windows" in SAS Companion for Windows
- "HEXw. Informat: z/OS" in SAS Companion for z/OS

**Syntax**

\[ \text{HEXw.} \]

**Syntax Description**

\( w \)

\( w \) 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 \( w < 16 \), HEXw converts the input value to positive integer binary values, treating all input values as positive (unsigned). If \( w = 16 \), HEXw 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 HEXw. informat ignores leading or trailing blanks.

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>88F 415200000000000000</td>
<td>2191 5.125</td>
</tr>
</tbody>
</table>

* The data line shows IBM mainframe hexadecimal data.

HHMMSSw. Informat

Reads hours, minutes, and seconds in the form `hh:mm:ss` or `hhmmss`.

**Syntax**

```
HHMMSSw.
```

**Syntax Description**

`w`

specifies the width of the input field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1–20</td>
</tr>
</tbody>
</table>

**Details**

The HHMMSSw. 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.

\texttt{mm}

is an integer that represents the number of minutes.

\texttt{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 seconds. 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.

\textbf{Example}

data test;
  time='12:45:44';
  sastime=input(time,hhmmss.);
  put sastime ;
  put sastime time8.;
run;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sastime ;</td>
<td>45944</td>
</tr>
</tbody>
</table>
### IBw.d Informat

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

**Categories:** 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:**
- “IB Informat: UNIX” in SAS Companion for UNIX Environments
- “IBw.d Informat: Windows” in SAS Companion for Windows
- “IBw.d Informat: z/OS” in SAS Companion for z/OS

## Syntax

### IBw.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 IBw.d informat reads integer binary (fixed-point) values, including negative values represented in two's complement notation. IBw.d reads integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

**Note:** Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering,
Comparisons

The IB\textsubscript{w.d} and PIB\textsubscript{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\textsubscript{w.d} and PIBR\textsubscript{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 462.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

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.

```sas
data one;
    result=input('0080'x,ib4.);
    put result;
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result on Big Endian Platforms</th>
<th>Result on Little Endian Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>put result;</td>
<td>128</td>
<td>-32768</td>
</tr>
</tbody>
</table>

See Also

Informats:
- “IBR\textsubscript{w.d} Informat” on page 559

**IBR\textsubscript{w.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\textsubscript{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 461.

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 462.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

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.

```sas
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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result on Big Endian Platforms</th>
<th>Result on Little Endian Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x;</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>put y;</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>result;</td>
<td>128</td>
<td>128</td>
</tr>
</tbody>
</table>

### See Also

**Informats:**
- “IBw.d Informat” on page 558

### 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**

**IEEEw.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 IEEEw.d informat is useful in operating environments where IEEE is the floating-point representation that is used. In addition, you can use the IEEEw.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\textit{w.d} informat allows other lengths, which enables you to read data from files that contain space-saving truncated data.

**Example**

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3F800000</td>
<td>1</td>
</tr>
<tr>
<td>3FF00000000</td>
<td>1</td>
</tr>
</tbody>
</table>

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

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

---

## JULIAN\textit{w}. Informat

Reads Julian dates in the form \textit{yyddd} or \textit{yyyyddd}.

**Categories:** CAS

Date and Time

**See:** “Julian Date Formats and Astronomical Dates” on page 9

**Syntax**

JULIAN\textit{w}.

**Syntax Description**

\textit{w}

specifies the width of the input field.

- **Default:** 5
- **Range:** 5–32

**Details**

The date values must be in one of these forms:

- \textit{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**

```sas
data test;
  date='2018099';
  sasdate=input(date,julian7.);
  put sasdate ;
  put sasdate date9.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate ;</td>
<td>21283</td>
</tr>
<tr>
<td>put sasdate date9.;</td>
<td>09APR2018</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

• “**JULIANw. Format**” on page 204

**Functions:**

• “**DATEJUL Function**” in *SAS Functions and CALL Routines: Reference*

• “**JULDATE Function**” in *SAS Functions and CALL Routines: Reference*

**System Options:**

• “**YEARCUTOFF= System Option**” in *SAS System Options: Reference*

---

**MDYAMPw.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.
Syntax

MDYAMPMw.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
caracters can be used as separators between any date or time component and
between the date and the time.

Comparisons

The MDYAMPMw.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 DATETIMEw.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 YMDDTTMw.d informat reads datetime values with required separators in the form
`<yy>yy-mm-dd/hh:mm:ss.ss`.

Example

```sas
data test;
  date='04-30-18 3:27 pm';
  sasdate=input(date,mdyampm.);
  put sasdate ;
  put sasdate datetime20. ;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate ;</td>
<td>1840721220</td>
</tr>
<tr>
<td>put sasdate datetime20. ;</td>
<td>30APR2018:15:27:00</td>
</tr>
</tbody>
</table>

See Also

Informats:
- “DATETIMEw. Informat” on page 538
- “YMDDTTMw.d Informat” on page 624

---

**MMDDYYw. Informat**

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

**Categories:**
- CAS
- Date and Time

**Syntax**

`MMDDYYw.`
**Syntax Description**

\( w \)

specifies the width of the input field.

Default: 6

Range: 6–32

**Details**

The date values must be in 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.

\( yy \text{ or } yyyy \)

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**

```sas
proc print data=test;
  date='05/04/2018';
  sasdate=input(date,mmddyy10.);
  put sasdate;
  put sasdate date9.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate ;</td>
<td>21308</td>
</tr>
<tr>
<td>put sasdate date9.;</td>
<td>04MAY2018</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “DATEw. Format” on page 150
- “DDMMYYw. Format” on page 156
MONYYw. Informat

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

Syntax

\texttt{MONYYw}.

\textbf{Syntax Description}

\textit{w}

\hspace{1cm} specifies the width of the input field.

\begin{tabular}{ll}
\textbf{Default} & 5 \\
\textbf{Range} & 5–32 \\
\end{tabular}

\textbf{Details}

The date values must be in one of these forms:

\begin{itemize}
\item \textit{mmmyy}
\item \textit{mmmyyyyy}
\end{itemize}

\textit{mmm}

\hspace{1cm} are the first three letters of the month name.

\textit{yy} or \textit{yyyy}

\hspace{1cm} is a two-digit or four-digit integer that represents the year.
A value read with the MONYYw. informat results in a SAS date value that corresponds to the first day of the specified month.

Note: SAS interprets a two-digit year as 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;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put sasdate ;</td>
<td>21305</td>
</tr>
<tr>
<td>put sasdate date9. ;</td>
<td>01MAY2018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DDMMYYw. Format” on page 156
- “MMDDYYw. Format” on page 207
- “MONYYw. Format” on page 218
- “YYMMDDw. Format” on page 430

Functions:
- “MONTH Function” in SAS Functions and CALL Routines: Reference
- “YEAR Function” in SAS Functions and CALL Routines: Reference

Informatst:
- “DDMMYYw. Informat” on page 540
- “MMDDYYw. Informat” on page 565
- “YYMMDDw. Informat” on page 626

System Options:
- “YEARCUTOFF= System Option” in SAS System Options: Reference

MSECw. Informat
Reads TIME MIC values.
Syntax

MSECw.

Syntax Description

w specifies the width of the input field.

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

Details

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

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

Comparisons

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

Example

    input btime msec8.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000EA043865A000</td>
<td>62818.412122</td>
</tr>
</tbody>
</table>

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

See Also

Informats:

- “TODSTAMPw. Informat” on page 612

NUMXw.d Informat

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

Categories: CAS, Numeric
Syntax

NUMXw.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 NUMXw.d informat reads numeric values and interprets a comma as a decimal point.

Comparisons

The NUMXw.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;

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

See Also

Formats:
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**

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

**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 OCTALw.d informat ignores leading and trailing blanks.

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

**Example**

```plaintext
data one;
  input @1 x octal13.1;
  put x;
datalines;
  177
; run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>177</td>
<td>12.7</td>
</tr>
</tbody>
</table>
**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 Informat: Windows” in SAS Companion for Windows
- “PDw.d Informat: z/OS” in SAS Companion for z/OS

**Syntax**

PD\_w\_d

**Syntax Description**

\( w \)

specifies the width of the input field.

**Default** 1  
**Range** 1–16

\( d \)

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

**Range** 0–10

**Details**

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

**Note:** Different operating environments store packed decimal values in different ways. However, PD\_w\_d reads packed decimal values with consistent results if the values are created 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

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

### Examples

#### Example 1: Reading Packed Decimal Data

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000128C</td>
<td>128</td>
</tr>
</tbody>
</table>

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

#### Example 2: Creating a SAS Date with Packed Decimal Data

```sas
input x: $hex10.;
mnth=input(x, pd5.);
date=input(put(mnth,8.),mmddyy6.);
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>012252010C</td>
<td>18621</td>
</tr>
</tbody>
</table>

* The data line is a hexadecimal representation of a binary number that is stored in packed decimal form in an IBM mainframe operating environment. Each byte occupies one column of the input field. The result is a SAS date value that corresponds to December 25, 2010.

---

**PDJULGw. Informat**

Reads packed Julian date values in the hexadecimal form yyyydddF for IBM.

- **Category:** Date and Time
- **Restriction:** This informat is not supported in a DATA step that runs in CAS.
Syntax
PDJULGw.

Syntax Description
w
specifies the width of the input field.

Default 4
Range 3–16

Details
The PDJULGw. informat reads IBM packed Julian date values in the form yyydddF:

yyy
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
input date pdjulg4.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012003F</td>
<td>18995</td>
</tr>
</tbody>
</table>

* SAS date value 18995 represents January 3, 2012.

See Also

Formats:
- “JULDAYw. Format” on page 202
- “JULIANw. Format” on page 204
- “PDJULGw. Format” on page 368
- “PDJUL1w. Format” on page 369

Functions:
- “DATEJUL Function” in SAS Functions and CALL Routines: Reference
- “JULDATE Function” in SAS Functions and CALL Routines: Reference
Informs:
- “JULIANw. Informat” on page 562
- “PDJULIw. Informat” on page 575

System Options:
- “YEARCUTOFF= System Option” in SAS System Options: Reference

PDJULIw. Informat
Reads packed Julian dates in the hexadecimal format ccyydddF for IBM.

Category: Date and Time
Restriction: This informat is not supported in a DATA step that runs in CAS.

Syntax
PDJULIw.

Syntax Description

w
specifies the width of the input field.

Default 4
Range 3–16

Details
The PDJULIw. informat reads IBM packed Julian date values in the form ccyydddF:

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 PDJULIw 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

input date pdjuli4.;
<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0099001F</td>
<td>14245</td>
</tr>
<tr>
<td>0112015F</td>
<td>19007</td>
</tr>
</tbody>
</table>

* SAS date value 14245 is January 1, 1999. SAS date value 19007 is January 15, 2012.

**See Also**

**Formats:**
- “JULDAYw. Format” on page 202
- “JULIANw. Format” on page 204
- “PDJULGw. Format” on page 368
- “PDJULIw. Format” on page 369

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

**Informats:**
- “JULIANw. Informat” on page 562
- “PDJULGw. Informat” on page 573

**System Options:**
- “YEARCUTOFF= System Option” in *SAS System Options: Reference*

---

**PDTIMEw. Informat**

Reads packed decimal time of SMF and RMF records.

**Categories:**
- CAS
- Date and Time

**Syntax**

PDTIMEw.

**Syntax Description**

w

specifies the width of the input field.
Requirement

\[ w \] must be 4 because packed decimal time values in SMF and RMF records contain 4 bytes of information.

Details

The PDTIME. 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 \(0\text{hhmmss}F\):

- 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. treats it as a missing value.

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

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0142225F</td>
<td>51745</td>
</tr>
</tbody>
</table>

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

**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**

```sas
data test;
  value='23.48%';
  percent=input(value,percent7.2);
  put percent= ;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>23.48%</td>
<td>0.2348</td>
</tr>
</tbody>
</table>

**PIB\( w.d \) 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 Informat: Windows” in SAS Companion for Windows

**Syntax**

\( \text{PIB}w.d \)
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\textsubscript{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 461.

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\textsubscript{w.d} informat treats all values as positive and includes the sign bit as part of the value.

• The PIB\textsubscript{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\textsubscript{w.d} and PIB\textsubscript{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\textsubscript{w.d} and PIBR\textsubscript{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 462.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

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.

```plaintext
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;  
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result on Big Endian Platforms</th>
<th>Result on Little Endian Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>put result;</td>
<td>193</td>
<td>65</td>
</tr>
<tr>
<td>put x;</td>
<td>256</td>
<td>1</td>
</tr>
<tr>
<td>put y;</td>
<td>1</td>
<td>256</td>
</tr>
</tbody>
</table>

**See Also**

**Informats:**

- “PIBRw.d Informat” on page 580

---

**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**

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

Note: Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 461.

Comparisons

• Positive integer binary values are the same as integer binary values, except that the sign bit is part of the value, which is always a positive integer. The PIBRw.d informat treats all values as positive and includes the sign bit as part of the value.

• The PIBRw.d informat with a width of 1 results in a value that corresponds to the binary equivalent of the contents of a byte. This is useful if your data 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 PIBw.d and PIBRw.d informats are equivalent.

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

To view a table that shows the type of informat to use with big endian and little endian integers, see “Reading Data Generated on Big Endian and Little Endian Platforms” on page 462.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

Example

```sas
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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result on Big Endian Platforms</th>
<th>Result on Little Endian Platforms</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x;</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>put y;</td>
<td>256</td>
<td>256</td>
</tr>
</tbody>
</table>
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**

`PKw.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 PKw.d informat is the same as the PDw.d informat, except that PKw.d treats the sign half of the field's last byte as part of the value, not as the sign of the value.

**Example**

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

**See Also**

Informat
- “PIBw.d Informat” on page 578
The data line is a hexadecimal representation of a binary number stored in unsigned packed decimal form. Each byte occupies one column of the input field.

**PUNCH.d Informat**

Reads whether a row of column-binary data is punched.

**Categories:**
- CAS
- Column Binary

**Syntax**

PUNCH.d

**Syntax Description**

`d`

specifies which row in a card column to read.

**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**

```sas
data one;
  input x punch.12;
  put x;
  datalines;
12-7-8
;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12-7-8</td>
<td>input x punch.12</td>
<td>1</td>
</tr>
</tbody>
</table>

* The data line is “virtual” punched card code. The punch card column for the example data has row 12 punched.
See Also

- “How to Read Column-Binary Data” in *SAS Language Reference: Concepts*

Informats:

- “$CBw. Informat” on page 492
- “CBw.d Informat” on page 533
- “ROWw.d Informat” on page 588

---

**RBw.d Informat**

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

**Syntax**

\[
\text{RB}\, w.d
\]

**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 RBw.d informat reads real binary values with consistent results if the values are created on the same type of operating environment that you use to run SAS.

**Comparisons**

The following table compares the names of real binary notation in several programming languages:
### Real Binary Notation

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>RB4.</td>
<td>RB8.</td>
</tr>
<tr>
<td>Fortran</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>double</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
</tbody>
</table>

**CAUTION:**

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

### Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128</td>
</tr>
</tbody>
</table>

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

### See Also

**Informats:**

- “IEEEw.d Informat” on page 561

---

**RMFDURw. Informat**

Reads duration intervals of RMF records.

**Categories:**

- CAS
- Date and Time

**Syntax**

```plaintext
RMFDURw.
```
Syntax Description

\[ w \]

specifies the width of the input field.

Requirement  \( w \) must be 4 because packed decimal duration values in RMF records contain 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 \( \text{mmsstt}F \):

- \( mm \) 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 \( \frac{1}{2} \)-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

```r
input dura rmfdur4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\text{----}</td>
<td>\text{---}</td>
</tr>
<tr>
<td>3552226F</td>
<td>2152.226</td>
</tr>
</tbody>
</table>

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

See Also

Informats:
RMFSTAMPw. Informat
Reads time and date fields of RMF records.

Categories:
- CAS
- Date and Time

Syntax
RMFSTAMPw.

Syntax Description

\( w \)

specifies the width of the input field.

| Requirement | \( w \) must be 8 because packed decimal time and date values in RMF records contain 8 bytes of information: 4 bytes of time data that are followed by 4 bytes of date data. |

Details
The RMFSTAMPw. informat reads packed decimal time and date values of RMF records that are produced by IBM mainframe systems. Then, the RMFSTAMPw. 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 \( 0\text{hhmmss}F\text{ccyyddd}F \):

| 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. |
| dddd | 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\textsubscript{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\textsubscript{w}. informat and the PDTIME\textsubscript{w}. informat read packed decimal values from RMF records. The RMFSTAMP\textsubscript{w}. informat reads both time and date values and results in a SAS datetime value. The PDTIME\textsubscript{w}. informat reads only time values and results in a SAS time value.

**Example**

```sas
data one;
  input x $hex16.;
  y=input(x,rmfstamp8.);
  put y ;
  put y datetime16.;
datalines;
0142225F2618200F
;
run;
```

<table>
<thead>
<tr>
<th>Data Line*</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put y ;</td>
<td>80739901345</td>
</tr>
<tr>
<td>put y datetime16.;</td>
<td>18JUL18:14:22:25</td>
</tr>
</tbody>
</table>

* The data line is a hexadecimal representation of a binary time and date value that is stored in packed decimal form as it would appear in an RMF record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to July 18, 2018, 2:22.25 PM.

**ROW\textsubscript{w}.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\textsubscript{w}.d`

**Syntax Description**

- **\textsubscript{w}**: specifies the row where the field begins.
- **\textsubscript{d}**: specifies the column 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**

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

<table>
<thead>
<tr>
<th>Data Line*</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td>1</td>
<td>00</td>
</tr>
<tr>
<td>04</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>.</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

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

**See Also**

- “How to Read Column-Binary Data” in *SAS Language Reference: Concepts*

**Informats:**

- “$CBw.Informat” on page 492
- “CBw.d Informat” on page 533
S370FFw.d Informat

Reads EBCDIC numeric data.

**Categories:**
- CAS
- Numeric

**Syntax**

S370FFw.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 S370FFw.d informat reads numeric data that is represented in EBCDIC and converts the data to native format. If EBCDIC is the native format, S370FFw.d performs no conversion.

S370FFw.d reads EBCDIC numeric values that are represented with 1 byte per digit. Use S370FFw.d in other operating environments to read numeric data from IBM mainframe files.

S370FFw.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. S370FFw.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 S370FFw.d informat performs the same role for numeric data that the $EBCDICw.d informat does for character data. That is, on an IBM mainframe system, S370FFw.d has the same effect as the standard w.d informat. On all other systems, using S370FFw.d is equivalent to using $EBCDICw.d as well as using the standard w.d informat.

**Example**

input @1 x s370ff3.;
The data lines 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.

### 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,
Comparisons

- If you use SAS on an IBM mainframe, S370FIBw.d and IBw.d are identical.
- S370FPIBw.d, S370FIBUw.d, and S370FIBw.d are used to read big endian integers in any operating environment.

To view a table that shows the type of informat to use with big endian and little endian integers, see “Reading Data Generated on Big Endian and Little Endian Platforms” on page 462.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

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.

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

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

See Also

Informats

- “S370FIBUw.d Informat” on page 592
- “S370FPIBw.d Informat” on page 596

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

S370FIBUw.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 461.

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 462.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

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.

```sas
data _null_
result1=input('7F'x,s370fibu1.);
result2=input('F6'x,s370fibu1.);
put result1 +1 result2;
```
run;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put result1;</td>
<td>result1=127</td>
</tr>
<tr>
<td>put result2;</td>
<td>result2=246</td>
</tr>
</tbody>
</table>

See Also

Informats:
- “S370FIBw.d Informat” on page 591
- “S370FPIBw.d Informat” on page 596

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<sub>w.d</sub>

**Syntax Description**

<sub>w</sub>
- Specifies the width of the input field.
  - **Default:** 1
  - **Range:** 1–16

<sub>d</sub>
- 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\textit{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\textit{w.d} and PD\textit{w.d} informats are identical.
- The following table compares the equivalent packed decimal notation by programming language:

<table>
<thead>
<tr>
<th>Language</th>
<th>Packed Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FPD4.</td>
</tr>
<tr>
<td>PL/I</td>
<td>FIXED DEC(7,0)</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-3 PIC 9(7)</td>
</tr>
<tr>
<td>assembler</td>
<td>PL4</td>
</tr>
</tbody>
</table>

### S370FPDU\textit{w.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**

\texttt{S370FPDUw.d}

**Syntax Description**

\textit{w} specifies the width of the input field.

Default 1

Range 1–16

\textit{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 S370FPDUw.d in other operating environments to read unsigned packed decimal data from IBM mainframe files.

Comparisons

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

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345F</td>
<td>12345</td>
</tr>
</tbody>
</table>

* The data line is a hexadecimal representation of a binary number that is stored in packed decimal form. Each two hexadecimal 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

S370FPIBw.d

Syntax Description

w

  specifies the width of the input field.

  Default: 4

  Range: 1–8
The `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. S370FPIBw.d reads integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Use S370FPIBw.d for positive integer binary data that 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 461.

## Comparisons

- If you use SAS on an IBM mainframe, S370FPIBw.d and PIBw.d are identical.
- S370FPIBw.d, S370FIBUw.d, and S370FIBw.d are used to read big endian integers in any operating environment.

To view a table that shows the type of informat to use with big endian and little endian integers, see “Reading Data Generated on Big Endian and Little Endian Platforms” on page 462.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 34.

## 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.

```sas
data _null_;  
  result=input('0100',s370fpib2.);  
  put result;  
run;
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put result;</td>
<td>12337</td>
</tr>
</tbody>
</table>

## See Also

**Informs:**

- “S370FIBw.d Informat” on page 591
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**

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

**Comparisons**

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

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FRB4</td>
<td>S370FRB8</td>
</tr>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
</tbody>
</table>
Real Binary Notation

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fortran</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-1</td>
<td>COMP-2</td>
</tr>
<tr>
<td>assembler</td>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>double</td>
</tr>
</tbody>
</table>

See Also

Informats:
- “RBw.d Informat” on page 584

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

S370FZDw.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\textsubscript{w}.\textsubscript{d} in other operating environments to read zoned decimal data from IBM mainframe files.

Comparisons
• If you use SAS on an IBM mainframe, S370FZD\textsubscript{w}.\textsubscript{d} and ZD\textsubscript{w}.\textsubscript{d} are identical.
• The following table shows the equivalent zoned decimal notation for several programming languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>Zoned Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FZD.3</td>
</tr>
<tr>
<td>PL/I</td>
<td>PICTURE'99T'</td>
</tr>
<tr>
<td>COBOL</td>
<td>PIC S9(3) DISPLAY</td>
</tr>
<tr>
<td>assembler</td>
<td>ZL3</td>
</tr>
</tbody>
</table>

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1F2C3</td>
<td>123</td>
</tr>
<tr>
<td>F1F2D3</td>
<td>-123</td>
</tr>
</tbody>
</table>

* The data line contains a hexadecimal representation of a binary number stored in zoned decimal format 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.

See Also

Informats:
• “ZD\textsubscript{w}.\textsubscript{d} Informat” on page 631

S370FZDB\textsubscript{w}.\textsubscript{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
input @1 x s370fzdb8.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>40404040F14040C0</td>
<td>1000</td>
</tr>
<tr>
<td>4040404040F1F2D3</td>
<td>–123</td>
</tr>
</tbody>
</table>

* The data lines 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.

S370FZDL\(w.d\) 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

S370FZDLw.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 S370FZDLw.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 S370FZDLw.d informat is equivalent to the COBOL notation PIC S9(n) DISPLAY SIGN LEADING, where n is the number of digits.

Example

input @1 x s370fzd13.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000001</td>
<td>1</td>
</tr>
<tr>
<td>C1F2F3</td>
<td>123</td>
</tr>
<tr>
<td>D1F2F3</td>
<td>-123</td>
</tr>
</tbody>
</table>

* The data lines contain a hexadecimal representation of a binary number stored in zoned decimal format 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.
S370FZDS\(w.d\) 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

\[ \text{S370FZDS} \!\!_{w.d} \]

### 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

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
The data line 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.

**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**

S370FZDTw.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 S370FZDTw.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 S370FZDTw.d informat is equivalent to the COBOL notation PIC S9\( (n) \) DISPLAY SIGN TRAILING SEPARATE, where \( n \) is the number of digits.
Example

    input @1 x s370fzd4.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
</tr>
<tr>
<td>F1F2F34E</td>
<td>123</td>
</tr>
<tr>
<td>F1F2F360</td>
<td>-123</td>
</tr>
</tbody>
</table>

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal format 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.

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

S370FZDUw.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 S370FZDUw.d in other operating environments to read unsigned zoned decimal data from IBM mainframe files.
Comparisons

- The S370FZDUw.d informat is similar to the S370FZDw.d informat, except that the S370FZDUw.d informat rejects all sign digits except F.

- The S370FZDUw.d informat is equivalent to the COBOL notation PIC 9(n) DISPLAY, where n is the number of digits.

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1F2F3</td>
<td>123</td>
</tr>
</tbody>
</table>

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal format 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.

SHRSTAMPw. Informat

Reads date and time values of SHR records.

**Categories:**
CAS
Date and Time

**Syntax**

`SHRSTAMPw.`

**Syntax Description**

w
- specifies the width of the input field.

**Requirement**
- w must be 8 because packed decimal date and time values in SHR records contain 8 bytes of information: 4 bytes of date data that are followed by 4 bytes of time data.

**Details**

The SHRSTAMPw. informat reads packed decimal date and time values of SHR records that are produced by IBM mainframe environments. Then, the SHRSTAMPw. 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 `ccyydddhhmmssth`. 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 SHRSTAMPw. 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;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put shrstamp ;</td>
<td>5006666435.8</td>
</tr>
<tr>
<td>put shrstamp datetime. ;</td>
<td>27AUG18:12:40:36</td>
</tr>
</tbody>
</table>

* The data line is a hexadecimal representation of a packed decimal date and time value that is stored as it would appear in an SHR record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to August 27, 2018 12:40:36.

SMFSTAMPw. Informat

Reads time and date values of SMF records.

Categories: CAS

Date and Time
Syntax

SMFSTAMPw.

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 SMFSTAMPw. informat reads integer binary time values and packed decimal date values of SMF records that are produced by IBM mainframe systems. Then, the SMFSTAMPw. informat converts the time and date values to SAS datetime values.

The date portion of an SMF record in hexadecimal notation is ccyydddF.

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 SMFSTAMPw. 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;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put y ;</td>
<td>1532016635</td>
</tr>
</tbody>
</table>

---
The data line is a hexadecimal representation of a binary time and date value that is stored as it would appear in an SMF record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to July 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:** Date and Time  
**Restriction:** This informat is not supported in a DATA step that runs in CAS.

**Syntax**

```
STIMERw.
```

**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`
\texttt{ss}

is an integer that represents the number of seconds.

\texttt{mm}

is an integer that represents the number of minutes.

\texttt{hh}

is an integer that represents the number of hours.

**Example**

```sas
data one;
  input test stimer10.;
  put 'test= ' test;
  datalines;
234
333.4444
1846989
83769.6
202;
run;
```

<table>
<thead>
<tr>
<th>test</th>
<th>234</th>
</tr>
</thead>
<tbody>
<tr>
<td>test</td>
<td>333.4444</td>
</tr>
<tr>
<td>test</td>
<td>1846989</td>
</tr>
<tr>
<td>test</td>
<td>83769.6</td>
</tr>
<tr>
<td>test</td>
<td>202</td>
</tr>
</tbody>
</table>

**TIMEw. Informat**

Reads hours, minutes, and seconds in the form \texttt{hh:mm:ss.<ss><AM | PM>}, where special characters such as the colon (:) or the period (.) are used to separate the hours, minutes, and seconds.

**Categories:**

CAS

Date and Time

**Syntax**

\texttt{TIMEw.}

**Syntax Description**

\texttt{w}

specifies the width of the input field.

Default \hspace{1cm} 8

Range \hspace{1cm} 5–32

**Details**

The \texttt{TIMEw.} informat reads SAS time values in the form \texttt{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**  
is an integer that represents the number of seconds, and if needed, tenths of a second. Seconds and tenths of a second must always be separated by a period.

**AM | PM**  
AM indicates time between 12:00 midnight and 11:59 in the morning. PM indicates time between 12:00 noon and 11:59 at night.

Separate **hh**, **mm**, and **ss** with a special character. When the period is used as the special character, the time is interpreted in the order hours, minutes, and seconds. For example, 23.22 is 23 hours and 22 minutes, not 23 minutes and 22 seconds, or 23 seconds and 22 tenths of a second.

If you do not enter a value for seconds, SAS assumes a value of 0.

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

### Example

```sas
data test;
  time='12:56';
  sastime=input(time,time.);
  put sastime ;
  put sastime time8.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>46560</td>
</tr>
<tr>
<td>put sastime ;</td>
<td>12:56:00</td>
</tr>
</tbody>
</table>

### See Also

**Formats:**
- “HHMMw.d Format” on page 195
- “HOURw.d Format” on page 197
- “MMSSw.d Format” on page 211
- “TIMEw.d Format” on page 402

**Functions:**
TODSTAMPw. Informat

Reads an 8-byte time-of-day stamp.

**Categories:**
- CAS
- Date and Time

**Syntax**

`TODSTAMPw.`

**Syntax Description**

`w`

specifies the width of the input field.

**Requirement**

`w` must be 8 because the OS TIME macro or the STCK instruction on IBM mainframes each returns an 8-byte value.

**Details**

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

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

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

**Example**

```sas
data one;
  date='D4E7F43876B80000'x;
  date2=input(date,todstamp8.);
  put date hex32. /
  date2 /
  date2 datetime20.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>D4E7F43876B80000</td>
</tr>
</tbody>
</table>
The data line is a hexadecimal representation of a binary, 8-byte time-of-day clock value. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to September 9 2018, 11:59:58.

**TRAILSGNw. Informat**

Reads a trailing plus (+) or minus (−) sign.

**Categories:**

CAS  
Numeric

**Syntax**

TRAILSGNw.

**Syntax Description**

w  
specifies the width of the input field.

**Default** 6  
**Range** 1–32

**Details**

If the data contains a decimal point, the TRAILSGN informat honors the number of decimal places that are in the input data. If the data contains a comma, the TRAILSGN informat reads the value, ignoring the comma.

**Example**

```
input x trailsgn8.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,000</td>
<td>1000</td>
</tr>
<tr>
<td>1+</td>
<td>1</td>
</tr>
<tr>
<td>1-</td>
<td>-1</td>
</tr>
</tbody>
</table>
TUw. Informat

Reads timer units.

**Categories:**
- CAS
- Date and Time

**Syntax**

TUw.

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

**Example**

```plaintext
input btime tu4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>1.2+</td>
<td>1.2</td>
</tr>
<tr>
<td>1.2-</td>
<td>-1.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>8FC7A9BC</td>
<td>62818.41563</td>
</tr>
</tbody>
</table>

* The data line 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.
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**

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

**See Also**

- “RBw.d Informat” on page 584

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\textsubscript{w.d}

**Syntax Description**

\textit{w}

specifies the width of the output field.

- Default: 1
- Range: 1–32

\textit{d}

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

---

**Details**

The VMSZN\textsubscript{w.d} informat is similar to the ZD\textsubscript{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\textsubscript{w.d} informat and the ZD\textsubscript{w.d} informat is in the special character used for the last digit. The following table shows the special characters used by the VMSZN\textsubscript{w.d} informat.

<table>
<thead>
<tr>
<th>Desired Digit</th>
<th>Special Character</th>
<th>Desired Digit</th>
<th>Special Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>-0</td>
<td>p</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>-2</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>-3</td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>-4</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>-5</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>-6</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>-7</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>-8</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>-9</td>
<td>y</td>
</tr>
</tbody>
</table>

---

**Example**

```plaintext
input @1 vmszn4.;
```
w.d Informat

Reads standard numeric data.

**Categories:**
- CAS
- Numeric

**Alias:**
- BESTw.d, Dw.d, Ew.d, Fw.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.

### Data line | Result
---|---
----|---
1234 | 1234
123t | -1234

**See Also**

**Formats:**
- “VMSZNw.d Format” on page 410

**Informats:**
- “ZDw.d Informat” on page 631
Comparisons

- The \texttt{w.d} informat is identical to the \texttt{BZw.d} informat, except that the \texttt{w.d} informat ignores trailing blanks in the numeric values. To read trailing blanks as 0s, use the \texttt{BZw.d} informat.
- The \texttt{w.d} informat can read values in scientific E notation exactly as the \texttt{Ew.d} informat does.

Example

```sas
data test;
  value='123.456';
  value2=input(value,8.3);
  put value2 ;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>123.456</td>
<td>123.456</td>
</tr>
</tbody>
</table>

\textbf{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.

\begin{tabular}{|l|l|}
\hline
\textbf{Category:} & Date and Time \\
\textbf{Restriction:} & This informat is not supported in a DATA step that runs in CAS. \\
\hline
\end{tabular}

\textbf{Syntax}

\texttt{WEEKUw.}

\textbf{Syntax Description}

- \texttt{W} specifies the width of the input field.
  
  \begin{tabular}{|l|l|}
  \hline
  \textbf{Default} & 11 \\
  \textbf{Range} & 3–200 \\
  \hline
  \end{tabular}

\textbf{Details}

The \texttt{WEEKUw.} 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 \texttt{WEEKUw.} uses the current year as the year expression, which is the default. If the input does not contain a day expression, then \texttt{WEEKUw.} 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 WEEKUw. informat are the same date for the following example. The current year is 2018.

<table>
<thead>
<tr>
<th>Width</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>3–4</td>
<td>www</td>
<td>w01</td>
</tr>
<tr>
<td>5–6</td>
<td>yyWww</td>
<td>18W01</td>
</tr>
<tr>
<td>7–8</td>
<td>yyWwwdd</td>
<td>18W0101</td>
</tr>
<tr>
<td>9–10</td>
<td>yyyyWwwdd</td>
<td>2018W0101</td>
</tr>
<tr>
<td>11–200</td>
<td>yyyy-Www-dd</td>
<td>2018-W01-01</td>
</tr>
</tbody>
</table>

**Comparisons**

The WEEKUw. 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 WEEKVw. 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 WEEKWw. 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.

```plaintext
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;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
</table>
### 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 and Time  
**Restriction:** This informat is not supported in a DATA step that runs in CAS.

#### Syntax

```
WEEKVw.
```

#### Syntax Description

- `w` specifies the width of the input field.

- **Default:** 11
- **Range:** 3–200
Details

The WEEKVw. informat reads the week-number value within a year. If the input does not contain a year expression, WEEKVw. uses the current year as the year expression, which is the default. If the input does not contain a day expression, WEEKVw. 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 WEEKVw. informat are the same date for the following example. The current year is 2018.

<table>
<thead>
<tr>
<th>Width</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>3–4</td>
<td>Www</td>
<td>w01</td>
</tr>
<tr>
<td>5–6</td>
<td>yyWww</td>
<td>18W01</td>
</tr>
<tr>
<td>7–8</td>
<td>yyWwwdd</td>
<td>18W0101</td>
</tr>
<tr>
<td>9–10</td>
<td>yyyyWwwdd</td>
<td>2018W0101</td>
</tr>
<tr>
<td>11–200</td>
<td>yyyy-Www-dd</td>
<td>2018–W01–01</td>
</tr>
</tbody>
</table>

Comparisons

The WEEKVw. 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 WEEKUw. 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 WEEKWw. 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;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put v;</td>
<td>21549</td>
</tr>
<tr>
<td>put w;</td>
<td>21185</td>
</tr>
<tr>
<td>put x;</td>
<td>21185</td>
</tr>
<tr>
<td>put y;</td>
<td>21185</td>
</tr>
<tr>
<td>put z;</td>
<td>21185</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WEEKUw. Format” on page 417
- “WEEKVw. Format” on page 419
- “WEEKWw. Format” on page 421

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

Informats:
- “WEEKUw. Informat” on page 618
- “WEEKWw. Informat” on page 622

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 and Time  
**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.
Details

The WEEKWw. informat reads the week-number value within the year. If the input does not contain a year expression, the WEEKWw. informat uses the current year as the year expression, which is the default. If the input does not contain a day expression, the WEEKWw. 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 WEEKWw. informat are the same date for the following example. The current year is 2018.

<table>
<thead>
<tr>
<th>Width</th>
<th>Format</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>3–4</td>
<td>Www</td>
<td>w01</td>
</tr>
<tr>
<td>5–6</td>
<td>yyWww</td>
<td>18W01</td>
</tr>
<tr>
<td>7–8</td>
<td>yyWwwdd</td>
<td>18W0101</td>
</tr>
<tr>
<td>9–10</td>
<td>yyyyWwwdd</td>
<td>2018W0101</td>
</tr>
<tr>
<td>11–200</td>
<td>yyyy-Www-dd</td>
<td>2018–W01–01</td>
</tr>
</tbody>
</table>

Comparisons

The WEEKWw. 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 WEEKUw. 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 WEEKVw. 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.

```sas
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;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put v;</td>
<td>21556</td>
</tr>
<tr>
<td>put w;</td>
<td>21185</td>
</tr>
<tr>
<td>put x;</td>
<td>21185</td>
</tr>
<tr>
<td>put y;</td>
<td>21185</td>
</tr>
<tr>
<td>put z;</td>
<td>21185</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WEEKUw. Format” on page 417
- “WEEKVw. Format” on page 419
- “WEEKWw. Format” on page 421

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

Informats:
- “WEEKUw. Informat” on page 618
- “WEEKVw. Informat” on page 620

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<,d

Syntax Description

\( w \)

specifies the width of the output field.

Default 19

Range 13–40

\( d \)

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

Default 0

Range 0–39

Details

The YMDDTTMw.d format reads SAS datetime values in the form \(<yy>yy-mm-dd\hh:mm:ss.ss>:\):

\( yy \) or \( yyyy \)

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

\( mm \)

is an integer between 01 and 12 that represents the month.

\( dd \)

is an integer between 01 and 31 that represents the day of the month.

\( hh \)

is an integer between 00 and 23 that represents hours.

\( mm \)

is an integer between 00 and 59 that represents minutes.

\( ss.ss \)

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

requirement If a fraction of a second is specified, the decimal point can be represented only by a period and is required.

- or : represents one of several special characters such as the slash (/), hyphen (-), colon (:), or a blank character that can be used to separate date and time components. Special characters can be used as separators between any date or time component and between the date and the time.

Comparisons

The YMDDTTM<,d informat reads datetime values with required separators in the form \(<yy>yy-mm-dd//hh:mm:ss.ss:>.\)
The MDYAMPMPw.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 DATETIMEw.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

```plaintext
data one;
  input dt ymddttm24.;
  put dt;
  datalines;
  2018-03-16 11:23:07.4
  2018 03 16 11 23 07.4
  18.3.16/11:23
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-03-16 11:23:07.4</td>
<td>1836818587.4</td>
</tr>
<tr>
<td>2018 03 16 11 23 07.4</td>
<td>1836818587.4</td>
</tr>
<tr>
<td>18.3.16/11:23</td>
<td>1836818580</td>
</tr>
</tbody>
</table>

### See Also

- “DATETIMEw. Informat” on page 538
- “MDYAMPMPw.d Informat” on page 563

---

**YYMMDDw. Informat**

Reads date values in the form *yyymmdd* or *yyyyyymmdd*.

**Categories:** CAS
Date and Time

**Syntax**

`YYMMDDw.`

**Syntax Description**

`w`

- specifies the width of the input field.
Details

SAS read date values in one of these forms:

- `yyymmdd`
- `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

```sas
data test;
  date='2018/05/04';
  sasdate=input(date,yyymmdd10.);
  put sasdate ;
  put sasdate date9.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018/05/04</td>
<td>21308 04MAY2018</td>
</tr>
</tbody>
</table>

See Also

**Formats:**

- “DATEw. Format” on page 150
- “DDMMYYw. Format” on page 156
- “MMDDYYw. Format” on page 207
- “YYMMDDw. Format” on page 430

**Functions:**
Informats:

- “DATEw. Informat” on page 537
- “DDMMYYw. Informat” on page 540
- “MMDDYYw. Informat” on page 565

System Options:

- “YEARCUTOFF= System Option” in SAS System Options: Reference

---

### YYMMNw. Informat

Reads date values in the form yyyymm or yymm.

**Categories:**
- CAS
  - Date and Time

**Syntax**

**YYMMNw.**

**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,yyymm6.);
  put sasdate ;
  put sasdate date9.;
runc;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>201805</td>
<td>21305</td>
</tr>
<tr>
<td></td>
<td>01MAY2018</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 150
- “DDMMYYw. Format” on page 156
- “YYMMDDw. Format” on page 430
- “YYMMw. Format” on page 429
- “YYMONw. Format” on page 436

Functions:
- “DAY Function” in SAS Functions and CALL Routines: Reference
- “MONTH Function” in SAS Functions and CALL Routines: Reference
- “MDY Function” in SAS Functions and CALL Routines: Reference
- “YEAR Function” in SAS Functions and CALL Routines: Reference

Informats:
- “DATEw. Informat” on page 537
- “DDMMYYw. Informat” on page 540
- “MMDDYYw. Informat” on page 565
- “YYMMDDw. Informat” on page 626

System Options:
- “YEARCUTOFF= System Option” in SAS System Options: Reference
**YYQw. Informat**

Reads quarters of the year in the form yyQq or yyyyQq.

**Categories:**
- CAS
- Date and Time

---

**Syntax**

`YYQw`.

**Syntax Description**

`w`

- specifies the width of the input field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>6 (For SAS version 6, the default is 4.)</td>
<td>4–32 (For SAS version 6, the range is 4–6.)</td>
</tr>
</tbody>
</table>

**Details**

SAS reads data in one of these forms:

- `yyQq`
- `yyyyQq`

- `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 `YYQw` 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**

```sas
data test;
  date='2018Q2';
  sasdate=input(date,yyq6.);
  put sasdate ;
  put sasdate date9.;
run;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>630</td>
<td>1</td>
</tr>
</tbody>
</table>
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 Informat: Windows” in SAS Companion for Windows
     “ZDw.d Format: z/OS” in SAS Companion for z/OS

Syntax

\[ ZDw.d \]

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 ZDw.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, ZDw.d reads zoned decimal values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

You can enter positive values in zoned decimal format from a 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 ZDw.d informat reads data in which every digit requires 1 byte. Use ZDVw.d or ZDw.d to read zoned decimal data in which the last byte contains the last digit and the sign.
- The ZDw.d informat functions like the ZDVw.d informat, with one exception: ZDVw.d validates the input string and disallows invalid data.
- The following table compares the zoned decimal informat with notation in several programming languages:

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

Example

    input @1 x zd4.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>F0F1F2C8</td>
<td>128</td>
</tr>
</tbody>
</table>

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

See Also

Informats

- “w.d Informat” on page 617
- “ZDVw.d Informat” on page 634
**ZDBw.d Informat**

Reads zoned decimal data in which zeros 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**

\texttt{ZDBw.d}

**Syntax Description**

\texttt{w}

specifies the width of the input field.

- Default: 1
- Range: 1–32

\texttt{d}

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

- Range: 0–31

**Details**

The ZDBw.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**

\begin{verbatim}
input @1 x zdb3.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>F140C2</td>
<td>102</td>
</tr>
</tbody>
</table>

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

Reads and validates zoned decimal data.

**Categories:**
- CAS
- Numeric

**Restriction:**
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**

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

**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 ZDVw.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.

ZDVw.d is dependent on the operating environment. For example, on IBM mainframes, the ZDVw.d informat requires an F for all high-order nibbles except the last. (In contrast, the ZDw.d informat ignores the high-order nibbles for all bytes except 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 ZDVw.d informat reads zoned decimal values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

**Comparisons**

The ZDVw.d informat functions like the ZDw.d informat, with one exception: ZDVw.d validates the input string and disallows invalid data.
Example

    input @1 test zdv4.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>F0F1F2C8</td>
<td>128</td>
</tr>
</tbody>
</table>

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

See Also

Informats:

- “w.d Informat” on page 617
- “ZDw.d Informat” on page 631
Recommended Reading

Here is the recommended reading list for this title:

- Base SAS Glossary
- Base SAS Procedures Guide
- SAS Companion for UNIX Environments
- SAS Companion for Windows
- SAS Companion for z/OS
- SAS DS2 Language Reference
- SAS Language Reference: Concepts
- The Little SAS Book, A Primer, Fifth Edition
- The SAS Workbook
- Step-by-Step Programming with Base SAS

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

SAS Books
SAS Campus Drive
Cary, NC 27513-2414
Phone: 1-800-727-0025
Fax: 1-919-677-4444
Email: sasbook@sas.com
Web address: sas.com/store/books
Index

Special Characters
SASCIiw. format 93
SASCIiw. informat 488
SBASE64Xw. format 94
SBASE64Xw. informat 489
SBINARYw. format 95
SBINARYw. informat 490
SCBw. informat 492
SCHARw. format 96
SCHARw. informat 493
  compared to $ASCII informat 489
  compared to $CHARZBw. informat 494
  compared to $EBCDICw. informat 495
  compared to $w. informat 509
SCHARZBw. informat 496
SFw. format 125
SFw. informat 508
SHEXw. format 100
SHEXw. informat 496
  compared to $BINARYw. informat 496
SMSGCASEw. format 101
SN8601BAw.d format 104
SN8601Bw.d format 102
SN8601Bw.d informat 497
SN8601EAw.d format 107
SN8601EHw.d format 109
SN8601Ew.d format 106
SN8601Ew.d informat 499
SN8601EXw.d format 111
SN8601Hw.d format 113
SN8601Xw.d format 115
SOCTALw. format 116
SOCTALw. informat 502
SPHEXw. informat 503
$QUOTEw. format 117
$QUOTEw. informat 504
SREVERJw. format 119
SREVERSw. format 120
SUPCASEw. format 121
SUPCASEw. informat 504
$UIDw. format 122
$VARYINGw. format 123
$VARYINGw. informat 506
Sw. format 125
Sw. informat 508
  compared to SCHARw. informat 493
%SYSFUNC function
  specifying formats with 6

A
AM or PM
datetime values with 151, 563
time values with 405
ANYDTEw. informat 509
ANYDTDMw. informat 512
ANYDTTMEw. informat 515
ASCII
  converting character data to 93
  ASCII data
  converting character data to, Base 64
    encoding 94, 489
  converting to native format 488
  astronomical date 9
ATTRIB statement
  specifying formats with 6
  specifying informats with 460
Australia
  monetary format 272, 309
Austria
  monetary format 319

B
B8601CI informat 517
B8601DAw. format 131
B8601DAw. informat 518
B8601DJ informat 520
B8601DNw. format 132
B8601DNw. informat 521
B8601DTw.d format 133
B8601DTw.d informat 523
B8601DXw. format 135
B8601DXw. informat
See B8601DZw.d informat
B8601DXw.d informat 525
B8601DZw. format 136
B8601DZw.d informat 525
B8601LXw. format 138
B8601LXw. informat
See B8601DTw.d informat
B8601LXw.d informat 525
B8601LZw. format 139
B8601LZw.d informat 527
B8601TXw. format 142
B8601TXw. informat
See B8601TZw.d informat
B8601TXw.d informat 528
B8601TZw.d format 143
B8601TZw.d informat 532
Base 64 encoding
converting character data to ASCII text 94, 489
BASE64X 489
Belgium
monetary format 282, 319
BESTDw.p format 128
BESTw. format 126
BESTw.d informat 617
big endian platforms
byte ordering 33
big endian platforms, byte ordering on 461
binary
converting character data to 95
converting numeric values to 130
binary data, converting to character 490
integers 530
binary zeros, converting to blanks 494
BINARYy.w. format 130
BINARYy.w.d informat 530
bits, extracting 531
BITSw.d informat 531
blanks
converting binary zeros to 494
converting to zeros 532
boundaries of intervals 27
Brazil
monetary format 274, 311
Bulgaria
monetary format 273, 310
byte ordering 33, 461
BZw.d informat 532
compared to w.d informat 618
C
calendar intervals, retail 31
Canada
monetary format 275, 312
CBw.d informat 533
character data
converting to ASCII 93
converting to ASCII text, Base 64 encoding 94, 489
converting to binary 95
converting to EBCDIC 98
converting to hexadecimal 100
converting to octal 116
converting to UUID 122
reverse order, left alignment 120
reverse order, preserving blanks 119
uppercase conversion 121
writing 96, 125
writing in uppercase 101
character data, reading
from column-binary files 492
standard format 508
varying length fields 506
with blanks 492
China
monetary format 277, 314
column-binary data, reading
down a column 588
punch-card code 583
column-binary files, reading 492
column-binary, reading
with blanks 493
commas
in numeric values 145, 147
replacing decimal points with 362
COMMAdw.d format 145
COMMAdw.d informat 535
compared to COMMAMxw.d informat 536
COMMAMxw.d format 147
COMMAMxw.d informat 536
compared to COMMAdw.d informat 535
Croatia
monetary format 285, 322
CSTR 97
custom intervals 31
Czech Republic
monetary format 278, 315
D
data conversion
formats and 35
data values, reading 457
date and time informats
name of month 231
quarter of the year 380
quarter of the year in Roman numerals 381
reading 22
tools by task 15
WEEKDATEw. format 413
WEEKDATXw. format 414
WEEKDAYw. format 416
WORDDATEw. format 423
WORDDATXw. format 424
writing 22
year 244
year 2000 and 13
year and name of month 236
year and quarter 240
year and week 245
year digits 12, 13
YEARw. format 427
YYMMDDw. format 430
YYMMDDxw. format 432
YYMNw. format 429
YYMNxw. format 434
YYMONw. format 436
YYQRw. format 441
YYQRxw. format 442
YYQw. format 438
YYQxw. format 439
date/time values, reading
date values, dddmmmyy 537
date values, dddmmmyy hh:mm:ss.ss 538
date values, dddmmmyyyy 537
date values, dddmmmyyyyy hh:mm:ss.ss 538
date values, ddmmyy 540
date values, ddmmyyyy 540
date, yymm 628
date, yymmm 628
dates, mmddyy 565
dates, mmddyyyy 565
dates, yymmdd 626
dates, yyyymmmdd 626
IBM mainframes 576
IBM mainframes, RMF records 587
IBM mainframes, SMF records 607
Julian dates 562
month and year values 567
RMF records 576
SMF records 576
TIME MIC values 568
time values, IBM mainframe 568
time-of-day stamp 612
time, hh:mm:ss.ss 610
timer units 614
year quarter 630
DATEAMPMw.d format 151
dates
date values as 224
datetime values
$N8601EA format, ISO 8601 extended notation 107
datetime formats
ISO 8601 extended datetime with no time zone 173
ISO 8601 extended datetime, with time zone 176
datetime informats
B8601DZ informat, ISO 8601 basic notation with time zone 525
datetime intervals 23
boundaries of 27
by category 24
multi-unit 28
multi-week 29
shifted intervals 30
single-unit 27
syntax 24
datetime values 9
$N8601 informat, ISO 8601 basic and extended notation 497
$N8601B format, basic notation 102
$N8601BA format, ISO 8601 basic notation 104
$N8601E format, extended notation 106
$N8601E informat, extended notation 499
$N8601EH format, ISO 8601 extended notation, hyphen for omitted components 109
$N8601EX format, extended notation, x for omitted components 111
$N8601H format, basic notation, hyphen for omitted components 113
$N8601X format, x for omitted components 115
as a datetime 246
as recognizable dates and times 21
B8601DN format, ISO 8601 basic datetime notation, formats the date 132
B8601DT format, ISO 8601 basic notation, no time zone 133, 135, 138
B8601DZ format, ISO 8601 basic notation with time zone 136
B8601TX format, ISO 8601 basic notation, time zone offset 142
DATEAMPMw.d format 151
DATETIMEw.d format 153
day of week 260
day of week and datetime 258
DTDATEw. format 164
DTMONYYw. format 165
DTWKDATXw. format 166
DTYEARw. format 168
DTYYYYCw. format 169
E8601DN format, ISO 8601 extended, formats the date 172
E8601DT format, ISO 8601 basic notation, no time zone 175, 178
E8601DT informat, ISO 8601 extended notation, no time zone 544
E8601DZ informat, ISO 8601 extended notation with time zone 546
E8601LZ informat, ISO 8601 extended local notation with time zone 548
E8601TX format, ISO 8601 basic notation, time zone offset 182
extracting from informat values 512
formats and informats and integrity of 14
name and day of month 251
name of month 255
time of day 256
tools by task 15
with a.m. or p.m. 247
with AM or PM 151
year 269
year 2000 and 13
year and name of month 261
year and name of week 270
year and quarter 265
year digits 12, 13
datetime values
YMDDTTMw.d informat 624
DATETIMEw. informat 538
DATETIMEw.d format 153
DATEw. format 150
DATEw. informat 537
DAYw. format 155
DDMMYYw. format 156
DDMMYYw. informat 540
DDMMYYxw. format 158
DEC format
integer binary (fixed-point) values in 200
positive integer binary (fixed-point) values in 376
reading integer binary values in 559
reading positive integer binary values in 580
decimal places
aligned 128, 148
decimal points
  replacing with commas 362
decimal points, reading as commas 569
Denmark
  monetary format 279, 316
DOLLARw.d format 160
DOLLARw.d informat 535
DOLLARXw.d format 161
DOLLARXw.d informat 536
double quotation marks
  data values in 117
DOWNAMEw. format 163
DTDATExw. format 164
DTMONYYxw. format 165
DTWKDATXw. format 166
DTYEARxw. format 168
DTYYQxw. format 169
duration values
  $N8601 informat, ISO 8601 basic and extended notation 497
duration 23
element 26
duration values
  $N8601B format, basic notation 102
  $N8601BA format, ISO 8601 basic notation 104
  $N8601E format, extended notation 106
  $N8601E informat, extended notation 499
  $N8601EA format, ISO 8601 extended notation 107
  $N8601EH format, ISO 8601 extended notation, hyphen for omitted components 109
  $N8601EX formats, extended notation, x for omitted components 111
  $N8601H format, basic notation, hyphen for omitted components 113
  $N8601X format, x for omitted components 115
Dw.d informat 617
Dw.p format 148
See E8601DZw.d informat
E8601Dxw.d informat 546
E8601Dzw. format 176
E8601Dzw.d informat 546
E8601Lxw. format 178
E8601Lxw. informat
  See E8601DTw.d informat
E8601Lxw.d informat 544
E8601Lzw. format 179
E8601Lzw.d informat 548
E8601TMxw.d format 181
E8601TMxw.d informat 550
E8601TXw. format 182
E8601TXw. informat
  See E8601TZw.d informat
E8601TXw.d informat 552
E8601TZw.d format 184
E8601TZw.d informat 552
EBCDIC
  converting character data to 98
  numeric data in 384
EBCDIC data
  convert to native format 495
  reading 590
Egypt
  monetary format 281, 318
  embedded characters, removing 535, 536
  encoding formats and 35
Estonia
  monetary format 280, 317
euros
  formats for 186, 189
EUROw.d format 186
EUROXw.d format 189
Ew. format 170
Ew.d informat 617

F
Faroe Island
  monetary format 279, 316
Finland
  monetary format 282, 319
fixed-point values
  DEC format 200, 376
  Intel format 200, 376
  reading in Intel and DEC formats 559, 580
  writing 198, 374
floating-point data (IEEE), reading 561
floating-point data, reading 554
floating-point values 191
IEEE 201
FLOATw.d format 191
FLOATw.d informat 554
FORMAT statement
  specifying formats with 6
  formats 4
  byte ordering and 33
  data conversions 35
  date values and 15
  datetime values and 15
  encodings 34
  packed decimal data 35
  permanent 7
  specifying with %SYSFUNC function 6
  specifying with ATTRIB statement 6
  specifying with FORMAT statement 6
  specifying with PUT function 5
  specifying with PUT statement 5
  syntax 4
  temporary 7
  time values and 15
  user-defined 7
  zoned decimal data 35
  formatting symbols
    ISO 8601 40, 467
    fractions 192, 425
    FRACTw. format 192
    France
      monetary format 282, 319
      Fw.d format 411
      Fw.d informat 617

G
  Germany
    monetary format 282, 319
  Greece
    monetary format 282, 319
  Greenland
    monetary format 279, 316

H
  HEX 100
  hexadecimal
    converting character data to 100
    converting real binary (floating-point) values to 193
    packed Julian dates in 368
    packed Julian dates in, for IBM 369
    hexadecimal binary values, converting to integers 555
    hexadecimal binary values, converting to real binary 555
    hexadecimal data, converting to character 496
    hexadecimal values
    reading packed Julian date values in, for IBM 573
    reading packed Julian dates in, for IBM 575
    HEXw. format 193
    HEXw. informat 555
    compared to $HEXw. informat 497
    HHMMSSw. informat 556
    HHMMw.d format 195
    Hong Kong
      monetary format 284, 321
    HOURw.d format 197
    Hungary
      monetary format 286, 323

I
  IBM
    packed Julian dates in hexadecimal for 369
    IBM dates and times, reading 517, 520
    IBM mainframe format
      integer binary (fixed-point) values in 385
      numeric data in 384
      packed decimal data in 389
      positive integer binary (fixed-point) values in 391
      real binary (floating-point) data in 393
      unsigned integer binary (fixed-point) values in 387
      unsigned packed decimal data in 390
      unsigned zoned decimal data in 400
      zoned decimal data 394
      zoned decimal leading-sign data in 396
      zoned decimal separate leading-sign data in 397
      zoned decimal separate trailing-sign data in 399
    IBM packed decimal data, reading 572
    IBRw.d format 200
    IBRw.d informat 559
    IBw.d format 198
    IBw.d informat 558
    compared to S370FIBw.d informat 592
    IEEE floating-point values 201
    reading 561
    IEEEw.d format 201
    IEEEw.d informat 561
    India
      monetary format 289, 326
    Indonesia
      monetary format 287, 324
    INFORMAT statement
      specifying informats with 459
      informats 457
byte ordering 461
date values and 15
datetime values and 15
integer binary notation 462
packed decimal data and 38
permanent 460
specifying, with ATTRIB statement 460
specifying, with INFORMAT statement 459
specifying, with INPUT function 459
specifying, with INPUT statement 459
syntax 458
temporary 460
time values and 15
user-defined 460
zoned decimal data and 38
INPUT function
specifying informats with 459
INPUT statement
specifying informats with 459
INTCK function
interval boundaries and 27
integer binary (fixed-point) values
IBM mainframe format 385
integer binary data
byte ordering 33
notation and programming languages 34
integer binary data, reading
IBM mainframe format 591, 596
integer binary notation 462
integer binary values
DEC format 200
Intel format 200
reading in Intel and DEC formats 559
writing 198
integer binary values, reading 558, 578
integers
printing without decimals 128
Intel format
integer binary (fixed-point) values in 200
positive integer binary (fixed-point) values in 376
reading integer binary values in 559
reading positive integer binary values in 580
interval values
$S8601 BA format, ISO 8601 basic notation, hyphen for omitted components 113
interval values
$S8601 informat, ISO 8601 basic and extended notation 497
$S8601B format, basic notation 102
$S8601E format, extended notation 106
$S8601E informat, extended notation 499
$S8601EH format, ISO 8601 extended notation, hyphen for omitted components 109
$S8601X format, x for omitted components 115
intervals 23
custom 31
date and time 23
ISO 8601 compliant 31
multi-unit 28
multi-week 29
retail calendar intervals 31
shifted 30
INTNX function
interval boundaries and 27
Ireland
monetary format 282, 319
IS8601DAw. format 171
IS8601DAw. informat 542
IS8601DNw. informat 543
IS8601DTw.d format 173
IS8601DTw.d informat 544
IS8601DZw. format 176
IS8601DZw.d informat 546
IS8601LZw. format 179
IS8601LZw.d informat 548
IS8601TMw.d format 181
IS8601TMw.d informat 550
IS8601TZw.d format 184
IS8601TZw.d informat 552
ISO 8601 31
ISO 8601 date and time formats
B8601DA format, basic date notation 131
B8601DN format, basic datetime notation, formats the date 132
B8601DT format, basic datetime notation, no time zone 133
B8601DX format, basic datetime notation, time zone offset 135
B8601DZ format, basic datetime notation with time zone 136
B8601LX format, basic datetime notation, time zone offset 138
B8601LZ format, basic local time with time zone 139
B8601TM format, basic time notation, no time zone 141
B8601TX format, basic time notation, time zone offset 142
B8601TZ format, basic time notation with time zone 143
E8601DA format, extended date notation 171
E8601DN format, extended datetime notation, formats the date 172
E8601DX format, basic datetime notation, time zone offset 175
E8601FX format, basic datetime notation, time zone offset 178
E8601TM format, extended time notation, no time zone 181
E8601TX format, basic time notation, time zone offset 182
E8601TZ format, extended time notation with time zone 184
extended datetime, with time zone 176
extended local time with UTC offset 179
ISO 8601 date and time informats
$N8601 informat, basic and extended notation for durations, datetimes, and intervals 497
$N8601E informat, extended notation for duration, datetime, and interval 499
B8601DA informat, basic date notation 518
B8601DN informat, basic datetime notation, returns the date in a datetime value 521
B8601DX informat, basic datetime notation with time zone 525
B8601TM informat, basic time notation, no time zone 527
B8601TZ informat, basic time notation with time zone 528
$N8601DA format, extended date notation 542
$N8601DN format, extended notation, returns date in datetime value 543
$N8601DT format, basic datetime notation, no time zone 544
$N8601DT format, extended datetime notation, no time zone 546
$N8601LZ format, extended local datetime notation with time zone 548
E8601TM informat, extended time notation, no time zone 550
E8601TZ informat, extended time notation with time zone 552
ISO 8601 datetime formats
extended datetime with no time zone 173
ISO 8601 duration and datetime formats
$N8601B format, basic notation 102
$N8601BA format, basic notation 104
$N8601E format, extended notation 106
$N8601EA format, extended notation 107
$N8601EH format, extended notation, hyphen for omitted components 109
$N8601H format, hyphen for omitted components 113
$N8601X format, x for omitted components 115
ISO 8601 formatting symbols 40, 467
ISO 8602 duration and datetime formats
$N8601EX formats, extended notation, x for omitted components 111
Israel
monetary format 288, 325
Italy
monetary format 282, 319

J
Japan
monetary format 290, 327
Japanese dates 221
Java dates and times, reading 520
JULDAYw. format 202
Julian date 9
Julian date values, packed
reading in hexadecimal form, for IBM 573
Julian dates 204, 464
day of the year 202
packed 36
packed values in hexadecimal 368
packed values in hexadecimal for IBM 369
Julian dates, packed
reading in hexadecimal format, for IBM 575
Julian format 9
JULIANw. format 204
JULIANw. informat 562

L
Latvia
monetary format 293, 330
leading zeros 450
Liechtenstein
monetary format 276, 313
Lithuania
monetary format 292, 329
little endian platforms
byte ordering 33
little endian platforms, byte ordering on 461
locale
best numerical notation based on 222
Luxembourg
monetary format 282, 319

M
Macau
monetary format 294, 331
Malaysia
monetary format 296, 333
Malta
monetary format 282, 319
MDYAMPMw.d format 205
MDYAMPMw.d informat 563
Mexico
monetary format 295, 332
MicroFocus COBOL
zoned numeric data 615
MicroFocus Cobol zoned numeric data 410
minus sign
trailing 613
MMDDYYw. format 207
MMDDYYw. informat 565
MMDDYYxw. format 209
MMSSw.d format 211
MMYYw. format 212
MMYYxw. format 214
monetary formats
Australia 272, 309
Austria 319
Belgium 282, 319
Brazil 274, 311
Bulgaria 273, 310
Canada 275, 312
China 277, 314
Croatia 285, 322
Czech Republic 278, 315
Denmark 279, 316
Egypt 281, 318
Estonia 280, 317
Faroe Island 279, 316
Finland 282, 319
France 282, 319
Germany 282, 319
Greece 282, 319
Greenland 279, 316
Hong Kong 284, 321
Hungary 286, 323
India 289, 326
Indonesia 287, 324
Ireland 282, 319
Israel 288, 325
Italy 282, 319
Japan 290, 327
Latvia 293, 330
Liechtenstein 276, 313
Lithuania 292, 329
Luxembourg 282, 319
Macau 294, 331
Malaysia 296, 333
Malta 282, 319
Mexico 295, 332
Netherlands 282, 319
New Zealand 298, 335
Norway 297, 334
Poland 299, 336
Portugal 282, 319
Puerto Rico 306, 343
Russia 300, 337
Singapore 302, 339
Slovenia 282, 319
South Africa 307, 344
South Korea 291, 328
Spain 282, 319
Sweden 301, 338
Switzerland 276, 313
Taiwan 305, 342
Thailand 303, 340
Turkey 304, 341
United Arab Emirates 271, 308
United Kingdom 283, 320
United States 306, 343
MONNAMEw. format 216
MONTHw. format 217
MONEYw. format 218
MONEYw. informat 567
MSEw. informat 568
multi-unit intervals 28
multi-week intervals 29

N
N8601B 102
N8601E 106
N8601EH 109
ND8601DAw. informat 518
ND8601DNw. informat 521
ND8601DTw.d informat 523
ND8601DZw.d informat 525
ND8601TMw.d informat 527
ND8601T Zw.d informat 528
negative numeric values
writing in parentheses 219
NEGPAREN Zw.d format 219
NENGOw. format 221
Netherlands
monetary format 282, 319
New Zealand
monetary format 298, 335
nibble 463
definition 36
NLBEST Zw. format 222
NLDATEL format 225
NLDATEM format 226
NLDATEMDL Format 228
NLDATEMDM format 229, 253
NLDATEMDS format 230
NLDATEMdw. format 227
NLDATEMnw. format 231
NLDATEMYL format 237
NLDATES format 232
NLDATEw. format 224
NLDATEWNw. format 234
NLDATEWw. format 233
NLDATEYMM format 238
NLDATEYMS format 239
NLDATEYmw. format 236
NLDATEYQL format 241
NLDATEYQM format 242
NLDATEYQS format 243
NLDATEYw. format 240
NLDATEYRw. format 244
NLDATEYWw. format 245
NLDATMAPw. format 247
NLDATMDTw. format 248
NLATML format 249
NLATMM format 250
NLATMMDL format 252
NLATMMDS format 254
NLATMMMDw. format 251
NLATMMNWw. format 255
NLATMS format 255
NLATMTMw. format 256
NLATMTZw. format
datetime values 258
NLATMw. format 246
NLATMWNw. format 260
NLATMWw. format 258
NLATMWZw. format
datetime values 260
NLATMYML format 262
NLATMYMM format 263
NLATMYMS format 264
NLATMYWw. format 261
NLATMYQL format 266
NLATMYQM format 267
NLDATMYQS format 268
NLDATMYQw. format 265
NLDATMYRw. format 269
NLDATMYWw. format 270
NLDATMZw. format
datetime values 270
NLMNIAEDw.d format 271
NLMNIAUDw.d format 272
NLMNIBGNw.d format 273
NLMNIBRLw.d format 274
NLMNIDCAdw.d format 275
NLMNICHFWw.d format 276
NLMNICYWw.d format 277
NLMNIZKWw.d format 278
NLMNIDDDw.d format 279
NLMNIEKw.d format 280
NLMNIEGpw.d format 281
NLMNIEURw.d format 282
NLMNIGBPw.d format 283
NLMNIHDKw.d format 284
NLMNIHMKw.d format 285
NLMNIIHFWw.d format 286
NLMNIIIDw.d format 287
NLMNIIISw.d format 288
NLMNIIINw.d format 289
NLMNIJPWw.d format 290
NLMNIKRw.d format 291
NLMNILTLw.d format 292
NLMNILVw.d format 293
NLMNINOPw.d format 294
NLMNIXNWw.d format 295
NLMNIMYRw.d format 296
NLMNINOKw.d format 297
NLMNINZDw.d format 298
NLMNIPNWw.d format 299
NLMNIRUw.d format 300
NLMNISEKWw.d format 301
NLMNISGDw.d format 302
NLMNITHBw.d format 303
NLMNITRYw.d format 304
NLMNITWDw.d format 305
NLMNIUSDw.d format 306
NLMNIZARw.d format 307
NLMNLAEDw.d format 308
NLMNLAUDw.d format 309
NLMNLBGNw.d format 310
NLMNLBRw.d format 311
NLMNLCDw.d format 312
NLMNLCFWw.d format 313
NLMNLNCYw.d format 314
NLMNLCZKw.d format 315
NLMNLDDKw.d format 316
NLMNLEEKw.d format 317
NLMNLEGpw.d format 318
NLMNLEURw.d format 319
NLMNLEURWw.d format 320
commas in 145, 147
converting to binary 130
converting to fractions 192
converting to octal 364
DOLLARw.d format 160
DOLLARXw.d format 161
Roman numerals 383
words with numeric fractions 425
writing as percentages 371
writing as words 426
writing negative values in parentheses 219
numerical notation
  best, based on locale 222
NUMXw.d format 362
NUMXw.d informat 569

O
  octal
    converting character data to 116
    converting numeric values to 364
  octal data
    converting to character 502
    converting to integers 571
  OCTALw. format 364
  OCTALw. informat
    compared to $OCTALw. informat 502
  OCTALw.d informat 571
  ODDSRw.d format 364

P
  p-values
    writing 378
  packed data, reading in IBM mainframe format 594
  packed decimal data 36, 464
defined 463
definition 35
  formats and 35
  formats and informats for 466
  IBM mainframe format 389
  languages supporting 37, 465
  platforms supporting 37, 465
  summary of formats and informats 38
  unsigned format 377
  packed decimal format
    writing data in 366
  packed hexadecimal data, converting to character 503
  packed Julian date values
    reading in hexadecimal form, for IBM 573
    writing in hexadecimal 368
    writing in hexadecimal for IBM 369
packed Julian dates 36, 464
reading in hexadecimal format, for IBM 575
parentheses
writing negative numeric values in 219
PDJULGw. format 368
PDJULGw. informat 573
PDJULIw. format 369
PDJULIw. informat 575
PDTIMEw. informat 576
compared to RMFSTAMPw. informat 588
PDw.d format 366
PDw.d informat 572
compared to SPHEXw. informat 503
compared to PKw.d informat 582
compared to S370FPDw.d informat 595
percentages
converting to numeric values 577
numeric values as 371
with minus sign for negative values 372
PERCENTNw.d format 372
PERCENTw.d format 371
PERCENTw.d informat 577
permanent formats 7
PIBRw.d format 376
PIBRw.d informat 580
PIBw.d format 374
PIBw.d informat 578
compared to S370FPIBw.d informat 597
PKw.d format 377
PKw.d informat 582
plus sign
trailing 613
PM or AM
datetime values with 151
time values with 405
Poland
monetary format 299, 336
Portugal
monetary format 282, 319
positive integer binary (fixed-point) values
IBM mainframe format 391
positive integer binary values
DEC format 376
Intel format 376
reading in Intel and DEC formats 580
writing 374
programming languages
integer binary notation and 34
packed decimal data support 37
zoned decimal data support 37
Puerto Rico
monetary format 306, 343
PUNCH.d informat 583
PUT function
specifying formats with 5
PUT statement
specifying formats with 5
PVALUEw.d format 378
Q
QTRRw. format 381
QTRw. format 380
quotation marks
removing 504
R
RBw.d format 382
RBw.d informat 584
compared to S370FRBw.d informat 598
compared to VAXRBw.d informat 615
reading data values 457
real binary (floating-point) data
IBM mainframe format 393
VMS format 409
real binary (floating-point) values
converting to hexadecimal 193
real binary data
real binary format 382
real binary data, reading 584
IBM mainframe format 598
VMS format 615
real binary format
real binary data (floating-point) in 382
retail calendar intervals 31
reverse order character data 119, 120
RMF records, reading duration intervals 585
RMFDURw. informat 585
RMFSTAMPw. informat 587
compared to RMFDURw. informat 586
roman numerals 441, 442
Roman numerals 381, 383
ROMANw. format 383
ROWw.d informat 588
Russia
monetary format 300, 337
S
S370FFw.d format 384
S370FFw.d informat 590
S370FIBUw.d format 387
S370FIBUw.d informat 592
S370FIBw.d format 385
S370FIBw.d informat 591
S370FPDUw.d format 390
S370FPDUw.d informat 595
S370FPDw.d format 389
S370FPDw.d informat 594
  compared to S370FPDUw.d informat 596
S370FPIBw.d format 391
S370FPIBw.d informat 596
  compared to S370FIBUw.d informat
S370FRBw.d format 393
S370FRBw.d informat 599
  compared to S370FZDUw.d informat 606
S370FZDB 600
S370FZDBw.d informat 600
S370FZDLw.d format 396
S370FZDLw.d informat 601
S370FZDSw.d format 397
S370FZDSw.d informat 603
S370FZDTw.d format 399
S370FZDTw.d informat 604
S370FZDUw.d format 400
S370FZDUw.d informat 605
S370FZDw.d format 394
S370FZDw.d informat 599
  compared to S370FZDUw.d informat 606
SAS informats 457
scientific notation 170
shifted intervals 30
SHR records
  reading data and time values of 606
SHRSTAMPw. informat 606
Singapore
  monetary format 302, 339
Slovenia
  monetary format 282, 319
SMFSTAMPw. informat 607
Social Security numbers 401
South Africa
  monetary format 307, 344
South Korea
  monetary format 291, 328
Spain
  monetary format 282, 319
SSNw. format 401
STIMERw. informat 609
Sweden
  monetary format 301, 338
Switzerland
  monetary format 276, 313

T
Taiwan
  monetary format 305, 342
temporary formats 7
Thailand
  monetary format 303, 340
time durations 23
time intervals 23
  boundaries of 27
  by category 24
  multi-unit 28
  shifted intervals 30
  single-unit 27
  syntax 24
time values 9
  as recognizable times 21
B8601LZ format, ISO 8601 basic local
time with time zone 139
B8601TM format, ISO 8601 basic
time notation, no time zone 141
B8601TM informat, ISO 8601 basic
time notation, no time zone 527
B8601TZ format, ISO 8601 basic
time notation with time zone 143
B8601TZ informat, ISO 8601 basic
time notation with time zone 528
E8601TM format, ISO 8601 extended
time notation, no time zone 181
E8601TM informat, ISO 8601 extended
time notation, no time zone 550
E8601TZ format, ISO 8601 extended
time notation with time zone 184
E8601TZ informat, ISO 8601 extended
time notation with time zone 552
extracting from informat values 515
formats and 15
HHMMw.d format 195
HOURw.d format 197
informats and 15
ISO 8601 extended local time with UTC
offset 179
MMSSw.d format 211
TIMEAMPMw.d format 405
TIMEw.d format 402
TODw.d format 406
tools by task 15
TIMEAMPMw.d format 405
TIMEw. informat 610
TIMEw.d format 402
TODSTAMPw. informat 612
  compared to MSECw. informat 569
TODw.d format 406
trailing plus or minus sign 613
TRAILSGNw. informat 613
transcoding 35
Turkey
  monetary format 304, 341
TUw. informat 614
U
United Arab Emirates monetary format 271, 308
United Kingdom monetary format 283, 320
United States monetary format 306, 343
unsigned integer binary (fixed-point) values
IBM mainframe format 387
unsigned integer binary data, reading
IBM mainframe format 592
unsigned packed decimal data
IBM mainframe format 390
unsigned packed decimal data, reading
582
IBM mainframe format 595
unsigned packed decimal format 377
unsigned zoned decimal data
IBM mainframe format 400
unsigned zoned decimal data, reading
IBM mainframe format 605
uppercase
$UPCASEw. informat 504
converting character data to 121
reading data as 504
writing character data in 101
user-defined formats 7
UUID
converting character data to 122

V
VAXRBw.d format 409
VAXRBw.d informat 615
VMS
zoned numeric data 410, 615
VMS format
real binary (floating-point) data in 409
VMSZNw.d format 410
VMSZNw.d informat 615

W
w.d format 411
w.d informat 617, 634
compared to NUMXw.d informat 570
compared to ZDw.d informat 632
WEEKDATEw. format 413
WEEKDATXw. format 414
WEEKDAYw. format 416
weeks
number of week, date value, U algorithm 618
week number, date value, V algorithm 620

week number, date value, W algorithm 622
week number, decimal format, U algorithm 417
week number, decimal format, V algorithm 419
week number, decimal format, W algorithm 421
WEEKU 417
WEEKUw. format 417
WEEKUw. informat 618
WEEKVw. format 419
WEEKVw. informat 620
WEEKWw. format 421
WEEKWw. inform 622
WORDDATEw. format 423
WORDDATXw. format 424
WORDFw. format 425
words
writing numeric values as 426
WORDSw. format 426
writing character data 96, 125

Y
year 2000 13
YEARCUTOFF= system option
year 2000 and 13
year digits and 12, 13
years
four-digit 12, 13
two-digit 12, 13
YEARw. format 427
YMDDTTMw.d finformat 624
YYMMDw. format 430
YYMMDw. informat 626
YYMMDxw. format 432
YYMMNw. inform 628
YYMMw. format 429
YYMMxw. format 434
YYMONw. format 436
YYQRw. format 441
YYQrw. format 442
YYQw. format 438
YYQw. informat 630
YYQxw. format 439
YYQzw. format 444
YYWEEKUw. format 446
YYWEEKVw. format 447
YYWEEKWw. format 449

Z
ZDBw.d informat 633
ZDVw.d informat 634
See also w.d informat
See also ZDw.d informat
compared to ZDw.d informat 632
ZDw.d format 451
ZDw.d informat 631, 634
  compared to ZDVw.d 634
zero
  numeric data with leading zeros 450
  zeros, binary
  converting to blanks 494
zoned decimal data 36, 464
  defined 463
  definition 36
  formats and 35
  formats and informats for 466
  IBM mainframe format 394
  languages supporting 37, 465
  platforms supporting 37, 465
  summary of formats and informats 38
  zoned decimal data, reading 631, 633
  IMB mainframe format 599
  zoned decimal format 451
  zoned decimal leading-sign data
    IBM mainframe format 396
    zoned decimal leading-sign data, reading
    IBM mainframe format 601
  zoned decimal separate leading-sign data
    IBM mainframe format 397
  zoned decimal separate trailing-sign data
    IBM mainframe format 399
  zoned decimal, separate trailing-sign data, reading
    IBM mainframe format 604
  zoned numeric data
    MicroFocus COBOL 410, 615
    VMS 410, 615
  zoned separate leading-sign data, reading
    IBM mainframe format 603
  Zw.d format 450
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore for additional books and resources.