SAS® 9.4 Formats and Informats Reference
Contents

About This Book ................................................................. v
What's New in SAS 9.4 Formats and Informats ......................... xi

PART 1  SAS Formats 1

Chapter 1 • About Formats ................................................ 3
   Definition of Formats ...................................................... 4
   Syntax ................................................................. 4
   Using Formats ......................................................... 5
   Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms . . . 8
   Data Conversions and Encodings ..................................... 9
   Working with Packed Decimal and Zoned Decimal Data ........... 10
   Working with Dates and Times By Using the ISO 8601 Basic and
      Extended Notations ............................................. 14

Chapter 2 • Dictionary of Formats ....................................... 29
   Formats Documented in Other Publications ....................... 31
   Formats by Category .................................................. 32
   Dictionary ............................................................. 42

PART 2  SAS Informats 225

Chapter 3 • About Informats ............................................. 227
   Definition of Informats .............................................. 227
   Syntax ................................................................. 228
   Using Informats ...................................................... 228
   Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms . . . 231
   Working with Packed Decimal and Zoned Decimal Data .......... 233
   Reading Dates and Times By Using the ISO 8601 Basic and Extended Notations . . . . 237

Chapter 4 • Dictionary of Informats ................................... 245
   Informats Documented in Other Publications .................... 247
   Informats by Category ............................................... 247
   Dictionary ............................................................. 254

Recommended Reading .................................................... 395
Index ........................................................................... 397
Contents
About This Book

Syntax Conventions for the SAS Language

Overview of Syntax Conventions for the SAS Language

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

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

Syntax Components

The components of the syntax for most language elements include a keyword and arguments. For some language elements, only a keyword is necessary. For other language elements, the keyword is followed by an equal sign (=). 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:
Some system options require that one of two keyword values be specified:

**DUPLEX | NODUPLEX**

Some procedure statements have multiple keywords throughout the statement syntax:

```sas
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:

```sas
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 and Multiple Arguments**

*argument-1=value-1 <argument-2=value-2 ...>*

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

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

```
LABEL variable-1=label-1 <variable-2=label-2 ...>;
```

*argument-1 <, argument-2, ...>*

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

Here are examples of this form of multiple arguments:

```
AUTHPROVIDERDOMAIN (provider-1:domain-1 <, provider-2:domain-2, ...>
INTO :macro-variable-specification-1 <, :macro-variable-specification-2, ...>
```

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

---

**Style Conventions**

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

**UPPERCASE BOLD**

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

```
ERROR <message>;
```

**UPPERCASE**

identifies arguments that are literals.

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

```
CMPMODEL= BOTH | CATALOG | XML |
```

**italic**

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

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

- nonliteral values that are assigned to an argument.

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

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

---

**Special Characters**

The syntax of SAS language elements can contain the following special characters:
an equal sign identifies a value for a literal in some language elements such as system options.

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

MAPS=location-of-maps

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

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

CAT (item-1 <, item-2, ...>)

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

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

CMPMODEL=BOTH | CATALOG | XML

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

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

CAT (item-1 <, item-2, ...>)

'value' or "value"

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

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

FOOTNOTE <n> <ods-format-options 'text' | "text">;

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

In this example, each statement ends with a semicolon:

data namegame;
  length color name $8;
  color = 'black';
  name = 'jack';
  game = trim(color) || name;
run;

References to SAS Libraries and External Files

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

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

```sas
infile file-specification obs = 100;
libname libref 'SAS-library';
```
What's New in SAS 9.4 Formats and Informats

Overview

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.

For SAS 9.4, there are no new or enhanced informats.

New SAS Formats

The following formats are new:

**B8601DXw.** (p. 75)
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 yyyyymmddThhmmss±hhmm.

**B8601LXw.** (p. 78)
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.

**B8601TXw.** (p. 82)
Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss±hhmm.

**E8601DXw.** (p. 112)
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-ddThh:mm:ss±hh:mm.

**E8601LXw.** (p. 115)
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.

**E8601TXw.** (p. 119)
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.
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 format might differ slightly from previous releases:

- BESTD<code>w.p</code>
- B8601DT<code>w.d</code>
- B8601TM<code>w.d</code>
- COMMAw<code>d</code>
- Dw<code>d</code>
- DATEAMPMw<code>d</code>
- DATETIMEw<code>d</code>
- DOLLARw<code>d</code>
- DOLLARXw<code>d</code>
- E8601DTw<code>d</code>
- E8601TMw<code>d</code>
- E8601DXw. on page 288
- B8601DXw. on page 289
- B8601DXw. on page 290
- B8601DXw. on page 292
- E8601DXw. on page 308
- E8601DXw. on page 310
- E8601DXw. on page 314

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

Enhancements to Existing Informats

In the second maintenance release for SAS 9.4, new aliases have been added for some time-zone informats:

<table>
<thead>
<tr>
<th>New Alias</th>
<th>Existing Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>B8601DXw. on page 288</td>
<td>B8601DZw.d on page 289</td>
</tr>
<tr>
<td>B8601LXw. on page 290</td>
<td>B8601DTw.d on page 287</td>
</tr>
<tr>
<td>B8601TXw. on page 292</td>
<td>B8601TZw.d on page 292</td>
</tr>
<tr>
<td>E8601DXw. on page 308</td>
<td>E8601DZw.d on page 309</td>
</tr>
<tr>
<td>E8601LXw. on page 310</td>
<td>E8601DTw.d on page 307</td>
</tr>
<tr>
<td>E8601TXw. on page 314</td>
<td>E8601TZw.d on page 314</td>
</tr>
</tbody>
</table>
Part 1

SAS Formats

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

Chapter 2
   Dictionary of Formats .................................................. 29
Chapter 1
About Formats

Definition of Formats .......................................................... 4
Syntax ................................................................................. 4
Using Formats ........................................................................ 5
  Ways to Specify Formats ..................................................... 5
  Permanent versus Temporary Association .............................. 6
  User-Defined Formats ........................................................ 7
  Restrictions on Formatting Dates ......................................... 7
Byte Ordering for Integer Binary Data on Big Endian and
Little Endian Platforms .......................................................... 8
  Definitions ................................................................. 8
  How Bytes Are Ordered Differently ..................................... 8
  Writing Data Generated on Big Endian or Little Endian Platforms ......................................................... 8
  Integer Binary Notation and Different Programming Languages .......................................................... 9
Data Conversions and Encodings .............................................. 9
Working with Packed Decimal and Zoned Decimal Data .............. 10
  Definitions ................................................................. 10
  Types of Data .............................................................. 11
  Platforms That Support Packed Decimal and Zoned Decimal Data .......................................................... 12
  Languages That Support Packed Decimal and Zoned Decimal Data ......................................................... 12
  Summary of Packed Decimal and Zoned Decimal Formats and Informats .................................................. 13
Working with Dates and Times By Using the ISO 8601 Basic
and Extended Notations ......................................................... 14
  ISO 8601 Formatting Symbols ........................................... 14
  Definitions ................................................................. 15
  About Dates and Times That Use the ISO 8601 Basic and Extended Notations ................................. 16
  Basic ISO 8601 Date, Time, and Datetime Values .................. 17
  Extended ISO 8601 Date, Time, and Datetime Values .................. 18
  Tips for Remembering UTC Formats That Use Time Zone Offsets ......................................................... 19
  Examples of Reading and Writing Basic and Extended ISO
    8601 Date, Time, and Datetime Values ................................ 20
  Writing ISO 8601 Duration, Datetime, and Interval Values ................................................................. 25
**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:

\[ <$> \text{format}<w>.<d> \]

- \$ 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” 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 commas, if any.

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

\begin{verbatim}
x=123;
put x= 2.;
\end{verbatim}
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 use formats in these ways:

- in a PUT statement
- with the PUT, PUTC, or PUTN functions
- with the %SYSFUNC macro function
- in a FORMAT statement in a DATA step or a PROC step
- in an ATTRIB statement in a DATA step or a PROC step

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

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

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

$1,145.32

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

PUT Function
The PUT function converts a numeric variable, a character variable, or a constant by using any valid format and then returns the resulting character value. For example, the following statement converts the value of a numeric variable to a two-character hexadecimal representation:

```sas
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, the following 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 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 Statements: Reference*.

**ATTRIB Statement**

The ATTRIB statement can also associate a format, as well as other attributes, with one or more variables. In the following 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 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 create your own formats. In Base SAS software, PROC FORMAT enables you to create your own formats for both character and numeric variables.

For more information, see “FORMAT” in *Base SAS Procedures Guide*.

When you execute a SAS program that uses user-defined formats, these formats can be made available in two ways:

- by creating permanent, not temporary, formats with PROC FORMAT
- by storing 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” in *Base SAS Procedures Guide*.

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

<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 and substitutes a default format, usually the BEST$w. or $w. format.</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 4000 might be off by days, depending on the computation.
Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms

Definitions

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

The ordering of bytes differs between the “big endian” and “little endian” platforms. These colloquial terms are used to describe byte ordering for IBM mainframes (big endian) and for platforms that are based on Intel (little endian). In the SAS System, the following platforms are considered big endian: AIX, HP-UX, IBM mainframe, Macintosh, and Solaris on SPARC. The following platforms are considered little endian: Intel ABI, Linux, OpenVMS Alpha, OpenVMS Integrity, Solaris on x64, Tru64 UNIX, and Windows.

How Bytes Are Ordered Differently

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

If an integer is negative, the two's complement representation is used. The high-order bit of the most significant byte of the integer is set on. For example, –2 would be represented in one, two, and four bytes on big endian platforms as FE, FF FE, and FF FF FF FE, respectively. On little endian platforms, the representation would be FE, FE FF, and FE FF FF FF. These representations result from the output of the integer binary value –2 expressed in hexadecimal notation.

Writing Data Generated on Big Endian or Little Endian Platforms

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

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

<table>
<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>
</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 where 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 one byte of storage. The last byte contains the number's sign as well as the last digit. Zoned decimal data produces a printable representation.

Nibble

specifies 1/2 of a byte.
Types of Data

Packed Decimal Data
A packed decimal representation stores decimal digits in each “nibble” of a byte. Each byte has two nibbles, and each nibble is indicated by a hexadecimal character. For example, the value 15 is stored in two nibbles, using the hexadecimal characters 1 and 5.

The sign indication is dependent on your operating environment. On IBM mainframes, the sign is indicated by the last nibble. With formats, C indicates a positive value, and D indicates a negative value. With informats, A, C, E, and F indicate positive values, and B and D indicate negative values. Any other nibble is invalid for signed packed decimal data. In all other operating environments, the sign is indicated in its own byte. If the high-order bit is 1, then the number is negative. Otherwise, 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 two 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 Informatns</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 $\text{ceil}((x+1)/2)$. For PIC 9(X) PACKED-DECIMAL, W is $\text{ceil}(x/2)$. For example, PIC S9(5) PACKED-DECIMAL represents five digits. If a sign is included, six nibbles are needed. $\text{ceil}((5+1)/2)$ has a length of three 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, and the Z directive indicates zoned decimal. 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</th>
<th>label</th>
<th>directive</th>
</tr>
</thead>
<tbody>
<tr>
<td>+000000</td>
<td>00001C</td>
<td>2</td>
<td>PEX1</td>
<td>DC PL3’1’</td>
</tr>
<tr>
<td>+000003</td>
<td>00001D</td>
<td>3</td>
<td>PEX2</td>
<td>DC PL3’-1’</td>
</tr>
<tr>
<td>+000006</td>
<td>F0F0C1</td>
<td>4</td>
<td>ZEX1</td>
<td>DC ZL3’1’</td>
</tr>
<tr>
<td>+000009</td>
<td>F0F0D1</td>
<td>5</td>
<td>ZEX2</td>
<td>DC ZL3’-1’</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. The formats and informs 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>Comments</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>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>
</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 ISO 8601 dates, time, datetime, durations, and interval values:

- $n$ specifies a number that represents the number of years, months, or days
- $P$ indicates that the duration that follows is specified by the number of years, months, days, hours, minutes, and seconds
- $T$ indicates that a time value follows. Any value with a time must begin with T.

---

<table>
<thead>
<tr>
<th>Format</th>
<th>Type of Data Representation</th>
<th>Corresponding Informat</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<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: $yyyydddFhhmmssth$, where $yyyyddd$ 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 $yyyyddd$ 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: $0hhmmsstttF$</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>RMFSTAMP</td>
<td>Input layout is: $0hhmmsstttFyyyydddF$, where $yyyyddd$ is the packed Julian date; $yyyy$ is a zero-based year from 1900</td>
</tr>
</tbody>
</table>
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/fffffff
specifies an optional fraction of a second using the digits 0–9:

/fff  use 1 - 3 digits for values read by the $N8601B informat and the $N8601E informat

/fffffff  use 1 - 6 digits for informat other than the $N8601B informat and the $N8601E informat

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  
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

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:

- 2013-02-01 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.
- 2013-02-01T12:52:34+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.
- P20130501T120000 is a duration value that is formatted using the basic notation.

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

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

- the zero meridian time or datetime near Greenwich, England (The offset is always +|–0000 or +|–00:00.)
- the local time or datetime, which uses the zero meridian time with a time zone offset for the local time
Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations

- a zero meridian datetime that uses a time zone offset for the user local time
- a user local time or datetime, which uses the current time for a time zone with a time zone offset for the user local time

**Basic ISO 8601 Date, Time, and Datetime Values**

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

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

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

<table>
<thead>
<tr>
<th>Date, Time, or Datetime</th>
<th>ISO 6801 Notation</th>
<th>Example</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>( yyyymmdd )</td>
<td>20120915</td>
<td>B8601DAw. on page 71</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writes the date from a</td>
<td>( yyyymmdd )</td>
<td>20120915</td>
<td>B8601DNw. on page 72</td>
</tr>
<tr>
<td>datetime.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>( hhmmss&lt;ffffff&gt; )</td>
<td>1553000322348</td>
<td>B8601TMw.d on page 81</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero meridian time and</td>
<td>( hhmmss+</td>
<td>–hhmm )</td>
<td>155300+0000</td>
</tr>
<tr>
<td>time zone offset.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The time zone offset is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>always +0000 or Z.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( hhmmssZ )</td>
<td>155300Z</td>
<td>B8601TZw. on page 83</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero meridian time that</td>
<td>( hhmmss+</td>
<td>–hhmm )</td>
<td>155300–0500</td>
</tr>
<tr>
<td>uses a time zone offset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>for the local time.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( hhmmss+</td>
<td>–hhmm )</td>
<td>155300–0500</td>
</tr>
<tr>
<td>Converts time to a user</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>local time by using a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time zone offset for the</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>user local time.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datetime</td>
<td>( yyyymmddThhmmss&lt;ffffff&gt; )</td>
<td>20120915T155300</td>
<td>B8601DTw.d on page 73</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero meridian datetime and</td>
<td>( yyyymmddThhmmss+</td>
<td>–hhmm )</td>
<td>20120915T155300–0000</td>
</tr>
<tr>
<td>time zone offset.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The time zone offset is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>always +0000 or Z.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>( yyyymmddThhmmssZ )</td>
<td>20120915T155300Z</td>
<td>B8601DZw. on page 76</td>
</tr>
</tbody>
</table>
### Date, Time, or Datetime

<table>
<thead>
<tr>
<th>ISO 6801 Notation</th>
<th>Example</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero meridian datetime that uses a time zone offset for the user local time.</td>
<td>yyyyymmddThhmmss+</td>
<td>--hhmm</td>
</tr>
<tr>
<td>Converts a datetime value to the user local time by using a time zone offset for the user local time.</td>
<td>yyyyymmddThhmmss+</td>
<td>--hhmm</td>
</tr>
</tbody>
</table>

---

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

---

### Extended ISO 8601 Date, Time, and Datetime Values

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

- **Date**: yyyy-mm-dd
- **Time**: hh:mm:ss.<ffffff>
- **Datetime**: yyyy-mm-ddThh:mm:ss.<ffffff>
- **Time with time zone**: hh:mm:ss.<ffffff>±|–hh:mm or hh:mm:ss.<ffffff>Z
- **Datetime with time zone**: yyyy-mm-ddThh:mm:ss.<ffffff>±|–hh:mm or yyyy-mm-ddTvh: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.

<table>
<thead>
<tr>
<th>Date, Time, or Datetime</th>
<th>ISO 6801 Notation</th>
<th>Example</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>yyyy-mm-dd</td>
<td>2012-09-15</td>
<td>E8601DAw. on page 108</td>
</tr>
<tr>
<td>Writes the date from a datetime.</td>
<td>yyyy-mm-dd</td>
<td>2012-09-15</td>
<td>E8601DNw. on page 72</td>
</tr>
<tr>
<td>Time</td>
<td>hh:mm:ss.&lt;ffffff&gt;</td>
<td>15:53:00.322348</td>
<td>E8601TMw.d on page 81</td>
</tr>
<tr>
<td>Zero meridian time and time zone offset.</td>
<td>hh:mm:ss.&lt;ffffff&gt;+</td>
<td>--hh:mm</td>
<td>15:53:00+00:00</td>
</tr>
<tr>
<td>The time zone offset is always +00:00.</td>
<td>hh:mm:ss.&lt;ffffff&gt;+</td>
<td>--hh:mm</td>
<td>15:53:00+05:00</td>
</tr>
<tr>
<td>Zero meridian time that uses a local time zone offset.</td>
<td>hh:mm:ss.&lt;ffffff&gt;+</td>
<td>--hh:mm</td>
<td>15:53:00+05:00</td>
</tr>
<tr>
<td>Converts time to user local time by using a local time zone offset.</td>
<td>hh:mm:ss+</td>
<td>--hh:mm</td>
<td>15:53:00+05:00</td>
</tr>
<tr>
<td>Datetime</td>
<td>yyyy-mm-ddTvh:mm:ss.&lt;ffffff&gt;</td>
<td>2012-09-15T15:53:00</td>
<td>E8601DTw.d on page 73</td>
</tr>
</tbody>
</table>
Zero meridian datetime and time zone offset.  
The time zone offset is always +00:00.

<table>
<thead>
<tr>
<th>Date, Time, or Datetime</th>
<th>ISO 8601 Notation</th>
<th>Example</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero meridian datetime that uses a time zone offset for the user local time.</td>
<td>yyyy-mm-ddTth:mm+</td>
<td>-zh:mm</td>
<td>2013-09-15T15:53:00+00:00</td>
</tr>
<tr>
<td>Converts a datetime value to the user local time by using a time zone offset for the user local time.</td>
<td>yyyy-mm-ddTth:mm+</td>
<td>-zh:mm</td>
<td>2013-09-15T185300–0500</td>
</tr>
</tbody>
</table>

An asterisk ( * ) is used in place of a date- or time-formatted value that is out-of-range. Increase 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></th>
<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>
<td></td>
</tr>
<tr>
<td>Zero meridian time that uses a local time zone offset</td>
<td></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 informat 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='15Sep2013:5:53:00'dt;
tm='05:53:00't;
put 'd=' d datetime.;
put 'e8601dz=' d e8601dz.;
put 'e8601lx=' d e8601lx.;
put 'e8601dx=' d e8601dx.;
put 'tm=' tm time.;
put 'e8601tz=' tm e8601tz.;
put 'e8601lz=' tm e8601lz.;
put 'e8601tx=' tm e8601tx.;
run;
```

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

- The output for the E8601DZ. and E8601TZ. formats write 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 write 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= 2013-09-15T05:53:00+00:00
e8601lx= 2013-09-15T05:53:00-05:00
```
Here are the results when the TIMEZONE= option is set to America/Los_Angeles:

```
d=15SEP13:05:53:00
```

• 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, 2012, and ends on April 8, 2012. The dates are recorded without time values, as follows: 20120402 and 2012-04-08. 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 output the newly created SAS dates in an easy-to-read layout rather than the numeric value of days since 1/1/1960.

```sas
data a;
input var1 b8601da8. +1 var2 e8601da10.;
put var1=b8601da. var2=e8601da.;
datalines;
20120402 2012-04-08;
run;
```

Here is the output from the SAS log:

```
var1=20120402 var2=2012-04-08
```

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:

```sas
data _null_;
input @1 dt b8601dn.;
put dt b8601dn.;
```
Here is the output from the SAS log:

**Reading and Writing Time Values with No Time Zone Offsets**
You can read time values that do not have time zone offset values into SAS time values by using the B8601TM.w.d and E8601TM.w.d informat. The B8601TM.w.d and the E8601TM.w.d formats write time values that do not have time zone offset values, as shown in this example:

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

Here is the output from the SAS log:

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

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

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

Here is the output from the SAS log:

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

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

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

Here is the output from the SAS log:

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

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

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

Here is the output from the SAS log:

x=13:49:02-04:00

**Reading and Writing Datetime Values with No Time Zone Offset**

In the following DATA step, SAS reads datetime values with the 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;  
20120402T124022 2012-04-02T12:30:22  
;  
run;

Here is the output from the SAS log:

dtB=20120402T124022 dtE=2012-04-02T12:30:22

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;  
20120402123245  
;  
run;

Here is the output from the SAS log:

dt1=20120402T123245

**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:

data _null_;  
x=input('2012-08-01T12:34:56-04:00',e8601dz25.);  
put x=e8601dz25.;  
run;

Here is the output from the SAS log:

x=2012-08-01T16:34:56+00:00

You use the B8601DXw. and E8601DXw. 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):

```
options timezone='europe/zurich';
data _null_;  
x=input('2012-08-01T12:34:56-04:00',e8601dz25.);
put x=e8601dx25.;
run;
```

Here is the output from the SAS log:

```
x=2012-08-01T18:34:56+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 E8601DXw. format. The time zone offset is the difference between the America/Anchorage time zone and UTC, which is 9 hours.

```
options timezone='america/anchorage';
data _null_;  
t='01Feb2013T12:34:56'dt;
put t=e8601dx.;
run;
```

Here is the output from the SAS log:

```
2013-02-01T03:34:56-09:00
```

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

Because time values are scalar, SAS does not typically compute time values based on the time zone of the programmer’s location. One exception to this rule occurs when a SAS time (not a datetime) is computed and then formatted with either the 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:

```
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. formats. This example writes the datetime for a user by using the time zone name PST (Pacific Time):

```
options timezone=pst;
data _null_;  
x='01Feb2013T12:34:56'dt;
put x=e8601lx.;
run;
```

Here is the output from the SAS log:

```
2013-02-01T12:34:56-07:00
```
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:

- \(PyyyyymmddThhmmss\)
- \(Pyyyy-mm-ddThh:mm:ss\)
- \(Pyyyy-mm-dd\) is a span of years, months, and days
- \(PnYnMnDTnHnMnSnS\) 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 \(PnW\) notation, \(W\) must be the only component in the duration.

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

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

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

<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>(PyyyyymmddThhmmssfff)</td>
<td>P20120915T155300</td>
<td>$N8601BA</td>
</tr>
<tr>
<td></td>
<td>(-PyyyyymmddThhmmssfff)</td>
<td>–P20120915T155300</td>
<td>$N8601BA</td>
</tr>
<tr>
<td>Duration - Extended Notation</td>
<td>(Pyyyy-mm-ddThh:mm:ss.fff)</td>
<td>P2012-09-15T15:53:00</td>
<td>$N8601EA</td>
</tr>
<tr>
<td></td>
<td>(-Pyyyy-mm-ddThh:mm:ss.fff)</td>
<td>–P2012-09-15T15:53:00</td>
<td>$N8601EA</td>
</tr>
<tr>
<td>Duration - Basic and Extended Notation</td>
<td>(PnYnMnDTnHnMnSnS)</td>
<td>P2y10m14dT20h13m45s</td>
<td>$N8601B $N8601E</td>
</tr>
<tr>
<td></td>
<td>(-PnYnMnDTnHnMnSnS)</td>
<td>–P2y10m14dT20h13m45s</td>
<td>$N8601B $N8601E</td>
</tr>
<tr>
<td>Time Component</td>
<td>ISO 8601 Notation</td>
<td>Example</td>
<td>Format</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------------</td>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>PrW (weeks)</td>
<td>P6w</td>
<td></td>
<td>$N8601B $N8601E</td>
</tr>
</tbody>
</table>

### Interval - Basic Notation

<table>
<thead>
<tr>
<th>yyyyymmddThhmmssfff</th>
<th>20120915T155300/20141113T000000</th>
<th>$N8601BA</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrYnMnDTnHnMnS/</td>
<td>P2y10M14dT20h13m45s/20120915T155300</td>
<td>$N8601B</td>
</tr>
<tr>
<td>yyyyymmddThhmmssfff</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrYnMnDTnHnMnS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Interval - Extended Notation

<table>
<thead>
<tr>
<th>yyyy-mm-ddT hh:mm:ss.fff</th>
<th>2012-09-15T15:53:00/2014-11-13T00:00:00</th>
<th>$N8601EA</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrYnMnDTnHnMnS/yyyy-mm-ddT hh:mm:ss.fff</td>
<td>P2y10M14dT20h13m45s/2012-09-15T15:53:00</td>
<td>$N8601E</td>
</tr>
<tr>
<td>yyyy-mm-ddT hh:mm:ss.fff</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrYnMnDTnHnMnS</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Datetime - Basic Notation

<table>
<thead>
<tr>
<th>yyyyymmddTHhhmmssfff+</th>
<th>-hhmm</th>
<th>20120915T155300</th>
<th>$N8601BA</th>
</tr>
</thead>
</table>

(all blank) $N8601B $N8601BA $N8601E $N8601EA

### Datetime - Extended Notation

<table>
<thead>
<tr>
<th>yyyy-mm-ddT hh:mm:ss.fff+</th>
<th>-hhmm</th>
<th>2012-09-15T15:53:00+04:30</th>
<th>$N8601EA</th>
</tr>
</thead>
</table>

(all blank) $N8601B $N8601BA $N8601E $N8601EA

---

**Writing Partial and Missing Components**

When any component of a date or time is not provided, it is called a partial value, and the components are considered missing. You can represent a missing component in a value by using a hyphen ( - ) or an x. A single hyphen represents the entire value for a given component. For example, one single hyphen can replace a four-digit year. A single x represents one character for a given component. A missing two-digit month would be written as xx. If the time portion is omitted when a date value is specified, the T must also be omitted.
Missing components in the durations form \(PnYnMnDTnHnMnS\) 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>$N8601H</td>
<td>yyyy-mm-ddThh:mm:ss</td>
<td>PnYnMnDTnHnMnS</td>
<td>--09-15T15:-:53 P2Y2DT4H5M6S/-09-15T15:-:00</td>
</tr>
<tr>
<td>$N8601EH</td>
<td>yyyy-mm-ddThh:mm:ss</td>
<td>Pyyyy-mm-ddThh:mm:ss</td>
<td>P0000---02T02:55:20/2012--15T---:--:45</td>
</tr>
<tr>
<td>$N8601X</td>
<td>yyyy-mm-ddThh:mm:ss</td>
<td>PnYnMnDTnHnMnS</td>
<td>P2Y2DT4H5M6S/xxxx-09-15T15:xx:00</td>
</tr>
<tr>
<td>$N8601EX</td>
<td>yyyy-mm-ddThh:mm:ss</td>
<td>Pyyyy-mm-ddThh:mm:ss</td>
<td>P0003-xx-02T02:55:20/2012-xx-15Txx:xx:45</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 2012---15 is written, and not 2012-15.

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, \(P2012---15\) is written, and not \(P2012-15\).

**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:

- \(p00030202T1031\)
- \(2012-09-15T15/2014-09-15T15:53\)
- \(--p0003-03-03T-:-:-\)
- \(P2y3m4dT5h6m\)
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>pt24h24m55s</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>p0003-02-61T15:61:61</td>
<td>P0003-04-01T16:02:01</td>
</tr>
<tr>
<td>p13m</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 Durations, Datetime, and Interval Values

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

• 2012-09-.xxTxx:xx:xx
• 2012

• P2012-09-15T10.33
• 2012-09-15/P0003-03-03,333
Chapter 2
Dictionary of Formats

Formats Documented in Other Publications ........................................... 31
Formats by Category ............................................................................. 32
Dictionary ............................................................................................. 42
$ASCIIw. Format ................................................................................ 42
$BASE64xw. Format ......................................................................... 43
$BINARYw. Format ........................................................................ 44
$CHARw. Format ............................................................................ 45
$EBCDICw. Format ....................................................................... 46
$HEXw. Format ............................................................................... 47
$MSGCASEw. Format .................................................................... 48
$N8601Bw.d Format ..................................................................... 49
$N8601BAw.d Format .................................................................... 50
$N8601Ew.d Format ..................................................................... 52
$N8601EAw.d Format .................................................................... 53
$N8601EHw.d Format .................................................................... 54
$N8601EXw.d Format .................................................................... 56
$N8601Hw.d Format ..................................................................... 57
$N8601Xw.d Format ..................................................................... 58
$OCTALw. Format ......................................................................... 59
$QUOTEw. Format ......................................................................... 60
$REVERSw. Format ...................................................................... 62
$UPCASEw. Format ....................................................................... 63
$VARYINGw. Format .................................................................... 64
$w. Format .................................................................................. 66
BESTw. Format ............................................................................. 67
BESTDw.p Format ......................................................................... 69
BINARYw. Format ......................................................................... 71
B8601DAw. Format ..................................................................... 71
B8601DNw. Format .................................................................... 72
B8601DTw.d Format .................................................................... 73
B8601DXw. Format .................................................................... 75
B8601DZw. Format ..................................................................... 76
B8601LXw. Format ..................................................................... 78
B8601LZw. Format ..................................................................... 79
B8601TMw.d Format .................................................................... 81
B8601TXw. Format ..................................................................... 82
B8601TZw. Format ..................................................................... 83
COMMAw.d Format ...................................................................... 85
COMMAXw.d Format .................................................................... 86
DATEAMPw.d Format ........................................ 89
DATEw. Format ............................................. 90
DATETIMEw.d Format ...................................... 92
DAYw. Format ............................................. 94
DDMMYYw. Format ......................................... 95
DDMMYYYw. Format ....................................... 96
DOLLARw.d Format ......................................... 98
DOLLARXw.d Format ....................................... 100
DOWNAMEw. Format ....................................... 101
DTDATEw. Format .......................................... 102
DTMONYYw. Format ....................................... 103
DTWكدXw. Format ......................................... 104
DTYEARw. Format ......................................... 106
DTYYQCw. Format ......................................... 107
Ew. Format .................................................. 108
E8601DAw. Format .......................................... 108
E8601DNw. Format .......................................... 109
E8601DTw.d Format ....................................... 110
E8601DXw. Format .......................................... 112
E8601DZw. Format .......................................... 113
E8601LXw. Format .......................................... 115
E8601LZw. Format .......................................... 116
E8601TMw.d Format ....................................... 118
E8601TXw. Format .......................................... 119
E8601TZw.d Format ....................................... 121
FLOATw.d Format ........................................... 122
FRACTw. Format ............................................ 124
HEXw. Format ............................................. 125
HHMMw.d Format .......................................... 126
HOURw.d Format ............................................ 128
IBw.d Format ............................................... 129
IBRw.d Format ............................................. 131
IEEEw.d Format ........................................... 132
JULDAYw. Format .......................................... 133
JULIANw. Format .......................................... 134
MDYAMPw.d Format ...................................... 135
MMDDYYw. Format ....................................... 137
MMDDYYXw. Format ...................................... 139
MMSSw.d Format ........................................... 141
MYYw. Format ............................................. 142
MYYXw. Format ............................................ 143
MONNAMEw. Format ...................................... 145
MONTHw. Format .......................................... 146
MONYYw. Format .......................................... 147
NEGPAREntw.d Format ................................... 148
NUMXw.d Format .......................................... 149
OCTALw. Format .......................................... 151
PDw.d Format ............................................. 151
PDJULGw. Format .......................................... 153
PDJULw. Format ............................................ 154
PERCENTw.d Format ...................................... 156
PERCENTNWd Format ..................................... 157
PiBw.d Format ............................................. 158
PIBRw.d Format ............................................ 160
PKw.d Format ............................................. 161
Additional formats are documented in these publications:

Formats by Category

There are four categories of formats in this list:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>
<tr>
<td>ISO 8601</td>
<td>instructs SAS to write date, time, and datetime values using the ISO 8601 standard.</td>
</tr>
<tr>
<td>Numeric</td>
<td>instructs SAS to write numeric data values from numeric variables.</td>
</tr>
</tbody>
</table>

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” 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>Character</td>
<td>$ASCII_w. Format (p. 42)</td>
<td>Converts native format character data to ASCII representation.</td>
</tr>
<tr>
<td></td>
<td>$BASE64Xw. Format (p. 43)</td>
<td>Converts character data into ASCII text by using Base 64 encoding.</td>
</tr>
<tr>
<td></td>
<td>$BINARYw. Format (p. 44)</td>
<td>Converts character data to binary representation.</td>
</tr>
<tr>
<td></td>
<td>$CHARw. Format (p. 45)</td>
<td>Writes standard character data.</td>
</tr>
<tr>
<td></td>
<td>$EBCDICw. Format (p. 46)</td>
<td>Converts native format character data to EBCDIC representation.</td>
</tr>
<tr>
<td></td>
<td>$HEXw. Format (p. 47)</td>
<td>Converts character data to hexadecimal representation.</td>
</tr>
<tr>
<td></td>
<td>$MSGCASEw. Format (p. 48)</td>
<td>Writes character data in uppercase when the MSGCASE system option is in effect.</td>
</tr>
<tr>
<td></td>
<td>$OCTALw. Format (p. 49)</td>
<td>Converts character data to octal representation.</td>
</tr>
<tr>
<td></td>
<td>$QUOTEw. Format (p. 60)</td>
<td>Writes data values that are enclosed in double quotation marks.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>$REVERJw. Format (p. 62)</td>
<td></td>
<td>Writes character data in reverse order and preserves blanks.</td>
</tr>
<tr>
<td>$REVERSw. Format (p. 63)</td>
<td></td>
<td>Writes character data in reverse order and left aligns</td>
</tr>
<tr>
<td>SUPCASEw. Format (p. 64)</td>
<td></td>
<td>Converts character data to uppercase.</td>
</tr>
<tr>
<td>$VARYINGw. Format (p. 64)</td>
<td></td>
<td>Writes character data of varying length.</td>
</tr>
<tr>
<td>$w. Format (p. 66)</td>
<td></td>
<td>Writes standard character data.</td>
</tr>
<tr>
<td>Date and Time</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SN8601Bw.d Format (p. 49)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyymmddThhmmss.</td>
</tr>
<tr>
<td>SN8601BAw.d Format (p. 50)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PyyyyymmdThhmmss and yyyymmddThhmmss.</td>
</tr>
<tr>
<td>SN8601Ew.d Format (p. 52)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss.</td>
</tr>
<tr>
<td>SN8601EAw.d Format (p. 53)</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.</td>
</tr>
<tr>
<td>SN8601EHw.d Format (p. 54)</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>SN8601EXw.d Format (p. 56)</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>SN8601Hw.d Format (p. 57)</td>
<td></td>
<td>Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss, dropping omitted components in duration values and using a hyphen (-) for omitted components in datetime values.</td>
</tr>
<tr>
<td>SN8601Xw.d Format (p. 58)</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>B8601DAw. Format (p. 71)</td>
<td></td>
<td>Writes date values by using the ISO 8601 basic notation yyyyymmdd.</td>
</tr>
<tr>
<td>B8601DNw. Format (p. 72)</td>
<td></td>
<td>Writes dates from datetime values by using the ISO 8601 basic notation yyyyymmdd.</td>
</tr>
<tr>
<td>B8601DTw.d Format (p. 73)</td>
<td></td>
<td>Writes datetime values by using the ISO 8601 basic notation yyyyymmddThhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>B8601DXw. Format (p. 75)</td>
<td></td>
<td>Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, writes the local date and time</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>by using the ISO 8601 datetime and time zone basic notation yyyymmddThhmmss+ hhmm.</td>
<td>B8601DZw. Format (p. 76)</td>
</tr>
<tr>
<td></td>
<td>by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation yyyymmddThhmmss+</td>
<td>–hhmm.</td>
</tr>
<tr>
<td></td>
<td>by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic time notation hhmmss+</td>
<td>–hhmm.</td>
</tr>
<tr>
<td></td>
<td>by using the ISO 8601 basic notation hhmmss&lt;fff&gt;.</td>
<td>B8601TMw.d Format (p. 81)</td>
</tr>
<tr>
<td></td>
<td>Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss+</td>
<td>–hhmm.</td>
</tr>
<tr>
<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+</td>
<td>–hhmm.</td>
</tr>
<tr>
<td></td>
<td>Writes date values in the form dddmmmyy, dddmmmyyyy, or ddm-mm-yyyy.</td>
<td>DATEw. Format (p. 89)</td>
</tr>
<tr>
<td></td>
<td>Writes datetime values in the form dddmmmyy:hh:mm:ss.ss with AM or PM.</td>
<td>DATEAMPMw.d Format (p. 90)</td>
</tr>
<tr>
<td></td>
<td>Writes datetime values in the form dddmmmyy:hh:mm:ss.ss.</td>
<td>DATETIMEw.d Format (p. 92)</td>
</tr>
<tr>
<td></td>
<td>Writes date values as the day of the month.</td>
<td>DAYw. Format (p. 94)</td>
</tr>
<tr>
<td></td>
<td>Writes date values in the form ddm&lt;yy&gt;yy or dd/mm/&lt;yy&gt;yy, where a forward slash is the separator and the year appears as either 2 or 4 digits.</td>
<td>DDMMYyw. Format (p. 95)</td>
</tr>
<tr>
<td></td>
<td>Writes date values in the form ddm&lt;yy&gt;yy or dd-mm-yy&lt;yy&gt;, where the x in the format name is a character that represents the special character that separates the day, month, and year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.</td>
<td>DDMMYYxw. Format (p. 96)</td>
</tr>
<tr>
<td></td>
<td>Writes date values as the name of the day of the week.</td>
<td>DOWNNAMEw. Format (p. 101)</td>
</tr>
<tr>
<td></td>
<td>Expects a datetime value as input and writes date values in the form ddmmyy or ddmmyyyy.</td>
<td>DTDATExw. Format (p. 102)</td>
</tr>
<tr>
<td></td>
<td>Writes the date part of a datetime value as the month and year in the form mmmm or mmmmyyy.</td>
<td>DTMONYYw. Format (p. 103)</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>DTWKDATX</td>
<td>Format (p. 104)</td>
<td>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).</td>
</tr>
<tr>
<td>DTYEAR</td>
<td>Format (p. 106)</td>
<td>Writes the date part of a datetime value as the year in the form yy or yyyy.</td>
</tr>
<tr>
<td>DTYYQC</td>
<td>Format (p. 107)</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>E8601DA</td>
<td>Format (p. 108)</td>
<td>Writes date values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DN</td>
<td>Format (p. 109)</td>
<td>Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DT</td>
<td>Format (p. 110)</td>
<td>Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.ffffff.</td>
</tr>
<tr>
<td>E8601DX</td>
<td>Format (p. 112)</td>
<td>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-ddThh:mm:ss+hh:mm.</td>
</tr>
<tr>
<td>E8601DZ</td>
<td>Format (p. 113)</td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00.</td>
</tr>
<tr>
<td>E8601LX</td>
<td>Format (p. 115)</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±hh:mm.</td>
</tr>
<tr>
<td>E8601LZ</td>
<td>Format (p. 116)</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±hh:mm.</td>
</tr>
<tr>
<td>E8601TM</td>
<td>Format (p. 118)</td>
<td>Writes time values by using the ISO 8601 extended notation hh:mm:ss.ffffff.</td>
</tr>
<tr>
<td>E8601TX</td>
<td>Format (p. 119)</td>
<td>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.</td>
</tr>
<tr>
<td>E8601TZ</td>
<td>Format (p. 121)</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;ffffff&gt;±hh:mm.</td>
</tr>
<tr>
<td>HHMM</td>
<td>Format (p. 126)</td>
<td>Writes time values as hours and minutes in the form hh:mm.</td>
</tr>
<tr>
<td>HOUR</td>
<td>Format (p. 128)</td>
<td>Writes time values as hours and decimal fractions of hours.</td>
</tr>
<tr>
<td>JULDAY</td>
<td>Format (p. 133)</td>
<td>Writes date values as the Julian day of the year.</td>
</tr>
<tr>
<td>JULIAN</td>
<td>Format (p. 134)</td>
<td>Writes date values as Julian dates in the form yydd or yyyydd.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>MDYAMPMw.d Format (p. 135)</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>MMDDYYw. Format (p. 137)</td>
<td>Writes date values in the form mm/dd&lt;yy&gt;yy or mm/dd/&lt;yy&gt;yy, where a forward slash is the separator and the year appears as either 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>MMDDYYxw. Format (p. 139)</td>
<td>Writes date values in the form mmdd&lt;yy&gt;yy or mm/dd/&lt;yy&gt;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 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>MMSSw.d Format (p. 141)</td>
<td>Writes time values as the number of minutes and seconds since midnight.</td>
<td></td>
</tr>
<tr>
<td>MMYYw. Format (p. 142)</td>
<td>Writes date values in the form mmM&lt;yy&gt;yy, where M is the separator and the year appears as either 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>MMYYxw. Format (p. 143)</td>
<td>Writes date values in the form mm&lt;yy&gt;yy or mm-&lt;yy&gt;yy, where the x in the format name is a character that represents the special character that separates the month and the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>MONNAMEw. Format (p. 145)</td>
<td>Writes date values as the name of the month.</td>
<td></td>
</tr>
<tr>
<td>MONTHw. Format (p. 146)</td>
<td>Writes date values as the month of the year.</td>
<td></td>
</tr>
<tr>
<td>MONYYw. Format (p. 147)</td>
<td>Writes date values as the month and the year in the form mmmmyy or mmmyyyy.</td>
<td></td>
</tr>
<tr>
<td>PDJULGw. Format (p. 153)</td>
<td>Writes packed Julian date values in the hexadecimal format yyyydddF for IBM.</td>
<td></td>
</tr>
<tr>
<td>PDJULIw. Format (p. 154)</td>
<td>Writes packed Julian date values in the hexadecimal format ccyydddF for IBM.</td>
<td></td>
</tr>
<tr>
<td>QTRw. Format (p. 164)</td>
<td>Writes date values as the quarter of the year.</td>
<td></td>
</tr>
<tr>
<td>QTRRw. Format (p. 165)</td>
<td>Writes date values as the quarter of the year in Roman numerals.</td>
<td></td>
</tr>
<tr>
<td>TIMEw.d Format (p. 184)</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. 186)</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. 187)</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. 193)</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>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>WEEKDATXw. Format (p. 195)</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. 196)</td>
<td>Writes date values as the day of the week.</td>
<td></td>
</tr>
<tr>
<td>WEEKUw. Format (p. 197)</td>
<td>Writes a week number in decimal format by using the U algorithm.</td>
<td></td>
</tr>
<tr>
<td>WEEKVw. Format (p. 199)</td>
<td>Writes a week number in decimal format by using the V algorithm.</td>
<td></td>
</tr>
<tr>
<td>WEEKWw. Format (p. 200)</td>
<td>Writes a week number in decimal format by using the W algorithm.</td>
<td></td>
</tr>
<tr>
<td>WORDDATEw. Format (p. 202)</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. 203)</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. 206)</td>
<td>Writes date values as the year.</td>
<td></td>
</tr>
<tr>
<td>YYMMw. Format (p. 207)</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 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>YYMMDDw. Format (p. 209)</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 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>YYMMDDxw. Format (p. 210)</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 which separates the year, month, and day. The special character can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>YYMONw. Format (p. 214)</td>
<td>Writes date values in the form yymmn or yyyyymmn.</td>
<td></td>
</tr>
<tr>
<td>YYQw. Format (p. 215)</td>
<td>Writes date values in the form &lt;yy&gt;yyQq, where Q is the separator, the year appears as either 2 or 4 digits, and q is the quarter of the year.</td>
<td></td>
</tr>
<tr>
<td>YYQxw. Format (p. 216)</td>
<td>Writes date values in the form &lt;yy&gt;yqq 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, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>YYQRw. Format (p. 218)</td>
<td>Writes date values in the form &lt;yy&gt;yyQr, where Q is the separator, the year appears as either 2 or 4 digits, and qr is the quarter of the year expressed in roman numerals.</td>
<td></td>
</tr>
</tbody>
</table>
| YYQRxw. Format (p. 219) | Writes date values in the form <yy>yqr or <yy>yy-qr, where the x in the format name is a character that represents the special character that separates the year and the quarter or the year, which
<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO 8601</td>
<td>SN8601Bw.d Format (p. 49)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyymmddThhmms.</td>
</tr>
<tr>
<td></td>
<td>SN8601BAw.d Format (p. 50)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PyyyyymmddThhmms and yyyymmddThhmms.</td>
</tr>
<tr>
<td></td>
<td>SN8601Ew.d Format (p. 52)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyy-mm-ddThh:mm:ss.</td>
</tr>
<tr>
<td></td>
<td>SN8601EAw.d Format (p. 53)</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.</td>
</tr>
<tr>
<td></td>
<td>SN8601EHw.d Format (p. 54)</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></td>
<td>SN8601EXw.d Format (p. 56)</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></td>
<td>SN8601Hw.d Format (p. 57)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss, dropping omitted components in duration values and using a hyphen (-) for omitted components in datetime values.</td>
</tr>
<tr>
<td></td>
<td>SN8601Xw.d Format (p. 58)</td>
<td>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.</td>
</tr>
<tr>
<td>B8601DAw. Format (p. 71)</td>
<td>Writes date values by using the ISO 8601 basic notation yyyymmdd.</td>
<td></td>
</tr>
<tr>
<td>B8601DNw. Format (p. 72)</td>
<td>Writes dates from datetime values by using the ISO 8601 basic notation yyyymmdd.</td>
<td></td>
</tr>
<tr>
<td>B8601DTw.d Format (p. 73)</td>
<td>Writes datetime values by using the ISO 8601 basic notation yyyymmddThhmms&lt;ffffff&gt;.</td>
<td></td>
</tr>
<tr>
<td>B8601DXw. Format (p. 75)</td>
<td>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 yyyymmddThhmms+hhmm.</td>
<td></td>
</tr>
<tr>
<td>B8601DZw. Format (p. 76)</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 yyyymmddThhmms+0000.</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>B8601LXw</td>
<td>Format (p. 78)</td>
<td>Writes datetimes as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation yyyyymmddThhmmss±</td>
</tr>
<tr>
<td>B8601LZw</td>
<td>Format (p. 79)</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±</td>
</tr>
<tr>
<td>B8601TMw</td>
<td>Format (p. 81)</td>
<td>Writes time values by using the ISO 8601 basic notation hhmmss&lt;fff&gt;.</td>
</tr>
<tr>
<td>B8601TXw</td>
<td>Format (p. 82)</td>
<td>Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, writes the local time by using the ISO 8601 basic time notation hhmmss±</td>
</tr>
<tr>
<td>B8601TZw</td>
<td>Format (p. 83)</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+</td>
</tr>
<tr>
<td>E8601DAw</td>
<td>Format (p. 108)</td>
<td>Writes date values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DNw</td>
<td>Format (p. 109)</td>
<td>Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DTw</td>
<td>Format (p. 110)</td>
<td>Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.</td>
</tr>
<tr>
<td>E8601DXw</td>
<td>Format (p. 112)</td>
<td>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-ddThh:mm:ss+</td>
</tr>
<tr>
<td>E8601DZw</td>
<td>Format (p. 113)</td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00.</td>
</tr>
<tr>
<td>E8601LXw</td>
<td>Format (p. 115)</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>
</tr>
<tr>
<td>E8601LZw</td>
<td>Format (p. 116)</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</td>
<td>Format (p. 118)</td>
<td>Writes time values by using the ISO 8601 extended notation hh:mm:ss&lt;fff&gt;.</td>
</tr>
<tr>
<td>E8601TXw</td>
<td>Format (p. 119)</td>
<td>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+</td>
</tr>
<tr>
<td>E8601TZw</td>
<td>Format (p. 121)</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>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Numeric</td>
<td>BESTw. Format (p. 67)</td>
<td>SAS chooses the best notation.</td>
</tr>
<tr>
<td></td>
<td>BESTD&lt;em&gt;w&lt;/em&gt;.p Format (p. 69)</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>BINARY&lt;em&gt;w&lt;/em&gt;. Format (p. 71)</td>
<td>Converts numeric values to binary representation.</td>
</tr>
<tr>
<td></td>
<td>COMMA&lt;em&gt;w&lt;/em&gt;.d Format (p. 85)</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>COMMAX&lt;em&gt;w&lt;/em&gt;.d Format (p. 86)</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&lt;em&gt;p&lt;/em&gt; Format (p. 87)</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></td>
<td>DOLLAR&lt;em&gt;w&lt;/em&gt;.d Format (p. 98)</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></td>
<td>DOLLARX&lt;em&gt;w&lt;/em&gt;.d Format (p. 100)</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></td>
<td>E&lt;em&gt;w&lt;/em&gt;. Format (p. 108)</td>
<td>Writes numeric values in scientific notation.</td>
</tr>
<tr>
<td></td>
<td>FLOAT&lt;em&gt;w&lt;/em&gt;.d Format (p. 122)</td>
<td>Generates a native single-precision, floating-point value by multiplying a number by 10 raised to the dth power.</td>
</tr>
<tr>
<td></td>
<td>FRACT&lt;em&gt;w&lt;/em&gt;. Format (p. 124)</td>
<td>Converts numeric values to fractions.</td>
</tr>
<tr>
<td></td>
<td>HEX&lt;em&gt;w&lt;/em&gt;. Format (p. 125)</td>
<td>Converts real binary (floating-point) values to hexadecimal representation.</td>
</tr>
<tr>
<td></td>
<td>IB&lt;em&gt;w&lt;/em&gt;.d Format (p. 129)</td>
<td>Writes native integer binary (fixed-point) values, including negative values.</td>
</tr>
<tr>
<td></td>
<td>IBR&lt;em&gt;w&lt;/em&gt;.d Format (p. 131)</td>
<td>Writes integer binary (fixed-point) values in Intel and DEC formats.</td>
</tr>
<tr>
<td></td>
<td>IEEE&lt;em&gt;w&lt;/em&gt;.d Format (p. 132)</td>
<td>Generates an IEEE floating-point value by multiplying a number by 10 raised to the dth power.</td>
</tr>
<tr>
<td></td>
<td>NEGPAREN&lt;em&gt;w&lt;/em&gt;.d Format (p. 148)</td>
<td>Writes negative numeric values in parentheses.</td>
</tr>
<tr>
<td></td>
<td>NUMX&lt;em&gt;w&lt;/em&gt;.d Format (p. 149)</td>
<td>Writes numeric values with a comma in place of the decimal point.</td>
</tr>
<tr>
<td></td>
<td>OCTAL&lt;em&gt;w&lt;/em&gt;. Format (p. 151)</td>
<td>Converts numeric values to octal representation.</td>
</tr>
<tr>
<td></td>
<td>PD&lt;em&gt;w&lt;/em&gt;.d Format (p. 151)</td>
<td>Writes data in packed decimal format.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>PERCENT</td>
<td>PERCENTw.d Format (p. 156)</td>
<td>Writes numeric values as percentages.</td>
</tr>
<tr>
<td>PERCENTN</td>
<td>PERCENTNw.d Format (p. 157)</td>
<td>Produces percentages, using a minus sign for negative values.</td>
</tr>
<tr>
<td>PIB</td>
<td>PIBw.d Format (p. 158)</td>
<td>Writes positive integer binary (fixed-point) values.</td>
</tr>
<tr>
<td>PIBR</td>
<td>PIBRw.d Format (p. 160)</td>
<td>Writes positive integer binary (fixed-point) values in Intel and DEC formats.</td>
</tr>
<tr>
<td>PK</td>
<td>PKw.d Format (p. 161)</td>
<td>Writes data in unsigned packed decimal format.</td>
</tr>
<tr>
<td>PVALUE</td>
<td>PVALUEw.d Format (p. 163)</td>
<td>Writes p-values.</td>
</tr>
<tr>
<td>RB</td>
<td>RBw.d Format (p. 165)</td>
<td>Writes real binary data (floating-point) in real binary format.</td>
</tr>
<tr>
<td>ROMAN</td>
<td>ROMANw. Format (p. 167)</td>
<td>Writes numeric values as roman numerals.</td>
</tr>
<tr>
<td>S370FF</td>
<td>S370FFw.d Format (p. 168)</td>
<td>Writes native standard numeric data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FIB</td>
<td>S370FIBw.d Format (p. 169)</td>
<td>Writes integer binary (fixed-point) values, including negative values, in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FIBU</td>
<td>S370FIBUw.d Format (p. 170)</td>
<td>Writes unsigned integer binary (fixed-point) values in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPD</td>
<td>S370FPDw.d Format (p. 172)</td>
<td>Writes packed decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPDU</td>
<td>S370FPDUw.d Format (p. 173)</td>
<td>Writes unsigned packed decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPIB</td>
<td>S370FPIBw.d Format (p. 174)</td>
<td>Writes positive integer binary (fixed-point) values in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FRB</td>
<td>S370FRBw.d Format (p. 176)</td>
<td>Writes real binary (floating-point) data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZD</td>
<td>S370FZDw.d Format (p. 177)</td>
<td>Writes zoned decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDL</td>
<td>S370FZDLw.d Format (p. 178)</td>
<td>Writes zoned decimal leading–sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDS</td>
<td>S370FZDSw.d Format (p. 179)</td>
<td>Writes zoned decimal separate leading-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDU</td>
<td>S370FZDUw.d Format (p. 182)</td>
<td>Writes unsigned zoned decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>SSN</td>
<td>SSNw. Format (p. 183)</td>
<td>Writes Social Security numbers.</td>
</tr>
<tr>
<td>VAXRB</td>
<td>VAXRBw.d Format (p. 189)</td>
<td>Writes real binary (floating-point) data in VMS format.</td>
</tr>
<tr>
<td>VMSZN</td>
<td>VMSZNw.d Format (p. 190)</td>
<td>Generates VMS and MicroFocus COBOL zoned numeric data.</td>
</tr>
</tbody>
</table>
### Dictionary

**$ASCIIw. Format**

Converts native format character data to ASCII representation.

- **Category**: Character
- **Alignment**: Left

**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**

```
put x $ascii3.;
```
The results are hexadecimal representations of ASCII codes for characters. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one character.

$BASE64Xw. Format

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

- **Category:** Character
- **Alignment:** Left

**Syntax**

$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 \((48+2)/3\times4=64\).

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 /.

The following are some uses of Base 64 encoding:

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

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

Example

```sas
put x $base64x64.;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;FCA01A7993BC&quot;</td>
<td>RkNBMDFBNzk5M0JD</td>
</tr>
<tr>
<td>&quot;MyPassword&quot;</td>
<td>TXlQYXNzd29yZA= =</td>
</tr>
<tr>
<td>&quot;www.mydomain.com/myhiddenURL&quot;</td>
<td>d3d3Lm15ZG9tYWluLmNvbi9teWhpZGRlbiVSTA= =</td>
</tr>
</tbody>
</table>

See Also

• The LIBNAME statement option “XMLDOUBLE=DISPLAY | INTERNAL” in SAS XML LIBNAME Engine: User’s Guide

Informats:

• “$BASE64Xw. Informat” on page 255

$\texttt{BINARYw. Format}

Converts character data to binary representation.

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

**Syntax**

$\texttt{BINARYw.}$

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

Example

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

<table>
<thead>
<tr>
<th>Value of name</th>
<th>ASCII</th>
<th>EBCDIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>0100000101000010</td>
<td>1100000111000010</td>
</tr>
</tbody>
</table>

$CHARw. Format

Writes standard character data.

- **Category:** Character
- **Alignment:** Left

Syntax

```
$CHARw.
```

**Syntax Description**

- **w** specifies the width of the output field.
  - **Default**: 8 if the length of variable is undefined; otherwise, the length of the variable
  - **Range**: 1–32767

Comparisons

- The $CHARw. format is identical to the $w. format.
- 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>Language</td>
<td>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>A(8)</td>
</tr>
</tbody>
</table>

**Example**

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

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>XYZ</td>
</tr>
</tbody>
</table>

### $EBCDICw. Format

Converts native format character data to EBCDIC representation.

- **Category:** Character
- **Alignment:** Left

#### Syntax

```plaintext
$EBCDICw:
```

#### Syntax Description

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

  - **Default:** 1
  - **Range:** 1–32767

#### Details

If EBCDIC is the native format, no conversion occurs.

On ASCII systems, the $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 the ENCODING, 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, $\text{EBCDIC}w$ converts ASCII character data to EBCDIC.
- On all other systems, $\text{EBCDIC}w$ behaves like the $\text{CHAR}w$ format.

**Example**

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

<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>

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

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

Converts character data to hexadecimal representation.

**Category:** Character  
**Alignment:** Left  
**See:** “$\text{HEX}w$. Format: UNIX” in *SAS Companion for UNIX Environments*  
“$\text{HEX}w$. Format: Windows” in *SAS Companion for Windows*

**Syntax**

$\text{HEX}w$

**Syntax Description**

`w` specifies the width of the output field.

**Default** The default width is calculated based on the length of the variable to be printed.
Range 1–32767

Tips To ensure that SAS writes the full hexadecimal equivalent of your data, make \( w \) twice the length of the variable or field that you want to represent. If \( w \) is greater than twice the length of the variable that you want to represent, $\text{HEX}w$. pads it with blanks.

Details
The $\text{HEX}w$. format converts each character to two hexadecimal characters. Each blank counts as one character, including trailing blanks.

Comparisons
The $\text{HEX}w$. format converts real binary numbers to their hexadecimal equivalent.

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

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>EBCDIC</td>
<td>ASCII</td>
</tr>
<tr>
<td>&quot;----&quot;</td>
<td>&quot;-----&quot;</td>
</tr>
<tr>
<td>AB</td>
<td>C1C2</td>
</tr>
</tbody>
</table>

$\text{MSGCASE}w$. Format
Writes character data in uppercase when the MSGCASE system option is in effect.

Category: Character
Alignment: Left

Syntax

$\text{MSGCASE}w$. 

Syntax Description

\( w \)
specifies the width of the output field.

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

Range 1–32767
Details

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

Example

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

<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*

---

### $N8601Bw.d Format

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

**Categories:** Date and Time

**ISO 8601:**

**Alignment:** Left

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

**Supports:** ISO 8601 Element 5.4.4, complete representation

**Syntax**

$N8601Bw.d$

**Syntax Description**

$w$

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1–200</td>
</tr>
<tr>
<td>Requirement</td>
<td>The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.</td>
</tr>
</tbody>
</table>
The $N8601B format writes ISO 8601 duration, datetime, and interval values as character data for the following basic notations:

- \(P^nY^nM^nDT^nH^nM^nS\)
- \(yyyymmddThhmmss\)
- \(P^nY^nM^nDT^nH^nM^nS/yyyymmddThhmmss\)
- \(yyyymmddThhmmssT/P^nY^nM^nDT^nH^nM^nS\)

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

- \(p2y3.5m\)
- \(p00020304T05.335\)

Example

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

<table>
<thead>
<tr>
<th>Value of nb</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002405050112FFC</td>
<td>P2Y4M5DT5H1M12S</td>
</tr>
<tr>
<td>2012915155300FFD</td>
<td>20120915T155300</td>
</tr>
<tr>
<td>2012915000000FFD2014915000000FFD</td>
<td>20120915T000000/20140915T000000</td>
</tr>
<tr>
<td>0033104030255FFC2012915155300FFD</td>
<td>P3Y1M4DT3H2M5SS/20120915T155300</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

$N8601BAw.d Format

Writes ISO 8601 duration, datetime, and interval forms by using the basic notations \(PyyyymmddThhmmss\) and \(yyyymmddThhmmss\).

- **Categories**: Date and Time
  ISO 8601
- **Alignment**: Left
- **Restriction**: 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.

<table>
<thead>
<tr>
<th>Default</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</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>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0–3</td>
</tr>
</tbody>
</table>

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

- \( PyyyymmddThhmmss \)
- \( yyyyymmddThhmmss \)
- \( PyyyymmddThhmmss/yyyyymmddThhmmss \)
- \( yyyyymmddThhmmss/PyyyyymmddThhmmss \)

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

- \( P00023.5 \)
- \( 00020304T05.335 \)

Example
put @1 nba $N8601ba.;

<table>
<thead>
<tr>
<th>Value of nba</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00024050501127D0</td>
<td>P0002405050112.7D0</td>
</tr>
<tr>
<td>2012915155300FFD</td>
<td>2012915155300FFD</td>
</tr>
<tr>
<td>00023040506075282012915155300FFD</td>
<td>P0002304050607.33/2012915155300FFD</td>
</tr>
</tbody>
</table>
See Also
“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

$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

Restriction: 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 0–3

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
Example

put @1 ne $n8601e.;

<table>
<thead>
<tr>
<th>Value of ne</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00024050501127D0</td>
<td>P2Y4M5DT5H1M12.5S</td>
</tr>
<tr>
<td>2012915155300FFD</td>
<td>2012-09-15T15:53:00</td>
</tr>
<tr>
<td>201291500000000000FFD</td>
<td>2012-09-15T00:00/2013-09-15T00:00:00</td>
</tr>
<tr>
<td>0033104030255FFC2012915155300FFD</td>
<td>P33Y1M4DT3H2M5SS/2012-09-15T15:53:00</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

$\texttt{N8601EAw.d Format}$

Writes ISO 8601 duration, datetime, and interval forms by using the extended notations $P\texttt{yyyy-mm-ddThh:mm:ss}$ and $\texttt{yyyy-mm-ddThh:mm:ss}$.

- **Categories**: Date and Time
- **ISO 8601**:
- **Alignment**: Left
- **Restriction**: UTC time zone offset values are not supported.
- **Supports**: ISO 8601 Element 5.4.4, complete representation

**Syntax**

$\texttt{N8601EAw.d}$

**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.
Details

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

- $Pyyyy-mm-dd$T$hh:mm:ss$
- $yyyy-mm-dd$T$hh:mm:ss$
- $Pyyyy-mm-ddT$hh:mm:ss/yyyy-mm-dd$T$hh:mm:ss$
- $yyyy-mm-ddT$hh:mm:ss/yyyy-mm-ddT$hh:mm:ss$

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

- $p00023.5$
- $0002–03–04T05.335$

Example

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

<table>
<thead>
<tr>
<th>Value of nea</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>000240505127D0</td>
<td>P0002–04–05T05:01:12.500</td>
</tr>
<tr>
<td>2012915155300FFD</td>
<td>2012–09–15T15:53:00</td>
</tr>
<tr>
<td>00023040506075282012915155300FFD</td>
<td>P0002–03–04T05:06:07.330/2012–09–15T15:53:00</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14
Syntax Description

\( w \)

specifies the width of the output field.

- **Default**: 50
- **Range**: 1–200
- **Requirement**: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

\( d \)

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

- **Default**: 0
- **Range**: 0–3

Details

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

- \( Pyyyy-mm-ddThh:mm:ss \)
- \( yyyy-mm-ddThh:mm:ss \)
- \( Pyyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss \)
- \( yyyy-mm-ddThh:mm:ss/Pyyyy-mm-ddThh:mm:ss \)
- \( yyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss \)

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

Example

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

<table>
<thead>
<tr>
<th>Value of ( a )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00023FFFFFFFFFFFFFC2012FFFF15FFFFFFFFD</td>
<td>P0002-03-T::-:/2012---T15::-</td>
</tr>
<tr>
<td>2012FFFF15FFFFFFFFFFFFF3FF1553FFFFFFFFC</td>
<td>2012--T15::-:/P-03--T15:53:-</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14
**$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.

<table>
<thead>
<tr>
<th>Categories:</th>
<th>Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ISO 8601</td>
</tr>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
<tr>
<td>Restriction:</td>
<td>UTC time zone offset values are not supported.</td>
</tr>
<tr>
<td>Supports:</td>
<td>ISO 8601 Elements 5.5.3, 5.5.4.1, and 5.5.4.2</td>
</tr>
</tbody>
</table>

**Syntax**

$N8601EXw.d$

**Syntax Description**

`w` specifies the width of the output field.

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

**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.

<table>
<thead>
<tr>
<th>Default</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0–3</td>
</tr>
</tbody>
</table>

**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.
Example

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

<table>
<thead>
<tr>
<th>Value of nex</th>
<th>Result</th>
</tr>
</thead>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

---

**$N8601Hw.d Format**

Writes ISO 8601 duration, datetime, and interval forms \( P_nY_nM_nD_nT_nH_nM_nS \) and \( yyyy-mm-ddT hh:mm:ss \), dropping omitted components in duration values and using a hyphen ( - ) for omitted components in datetime values.

- **Categories:** Date and Time
- **Alignment:** Left
- **Restriction:** UTC time zone offset values are not supported.
- **Supports:** ISO 8601 Elements 5.5.3, 5.5.4.1, and 5.5.4.2

### Syntax

\( $N8601H w.d \)

**Syntax Description**

\( w \)

- specifies the width of the output field.

  - **Default:** 50
  - **Range:** 1–200
  - **Requirement:** The minimum length for a duration value or a datetime value is 16.
    The minimum length for an interval value is 16.

\( d \)

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

  - **Default:** 0
  - **Range:** 0–3
Details

The $N8601H format writes ISO 8601 durations, intervals, and datetimes in the following forms, omitting components in the $nYnMnDTnHnMnS form and using a hyphen ( - ) to represent omitted components in the datetime form:

- \( P^nY^nM^nDT^nH^nM^nS \)
- yyyy-mm-ddThh:mm:ss
- PnYnMnDTnHnMnS/yyyy-mm-ddThh:mm:ss
- yyyy-mm-ddThh:mm:ss/PnYnMnDTnHnMnS
- yyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss

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

Example

put nh $n8601h.;

<table>
<thead>
<tr>
<th>Value of nh</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002304FFFFFFFFFCD2012FFP15FFFFFFFFFDD</td>
<td>P2Y3M4D/2012—T15:-:-</td>
</tr>
<tr>
<td>FPPF102FFFFFFFFFDD2012FFP15FFFFFFFFFDD</td>
<td>—01-02T-:-:-0/2012—T15:-:-</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

$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

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Elements 5.5.3, 5.5.4.1, and 5.5.4.2

Syntax

SN8601Xw.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 0–3

Details

The $N8601X$ format writes ISO 8601 durations, intervals, and datetimes in the following forms, omitting components in the $P_{n}Y_{n}M_{n}DT_{n}H_{n}M_{n}S$ form and using an $x$ to represent omitted components in the datetime form:

- $P_{n}Y_{n}M_{n}DT_{n}H_{n}M_{n}S$
- $yyyy-mm-ddThh:mm:ss$
- $P_{n}Y_{n}M_{n}DT_{n}H_{n}M_{n}S/yyyy-mm-ddThh:mm:ss$
- $yyyy-mm-ddThh:mm:ss/T/P_{n}Y_{n}M_{n}DT_{n}H_{n}M_{n}S$
- $yyyy-mm-ddThh:mm:ss/yyyy-mm-ddThh:mm:ss$

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

Example

`put nx $n8601x.;`

<table>
<thead>
<tr>
<th>Value of nx</th>
<th>Result</th>
</tr>
</thead>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

$OCTALw. Format$

Converts character data to octal representation.

Category: Character

Alignment: Left
Syntax

\texttt{SOCTAL\textsubscript{\textit{w}}.}

**Syntax Description**

\textit{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 \textit{w} by three times the length of the character value.

**Comparisons**
The \texttt{SOCTAL\textsubscript{\textit{w}}.} format converts character values to the octal representation of their character codes. The \texttt{OCTAL\textsubscript{\textit{w}}.} format converts numeric values to octal representation.

**Example**
The following example shows ASCII output when you use the \texttt{SOCTAL\textsubscript{\textit{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.

```
141162164040040  
162151143145040  
14214156153040  
```

**$QUOTE\textsubscript{\textit{w}}. Format**

Writes data values that are enclosed in double quotation marks.

**Category:** Character

**Alignment:** Left

**Syntax**

\texttt{QUOTE\textsubscript{\textit{w}}.}
Syntax Description

\[ w \]

specifies the width of the output field.

Default

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

Range

2–32767

Tip

Make \( w \) wide enough to include the left and right quotation marks.

Details

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

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

Example

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

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-------</td>
</tr>
<tr>
<td>Value of name</td>
<td>Result</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------</td>
</tr>
<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''verb''</td>
<td>&quot;'ad''verb''&quot;</td>
</tr>
<tr>
<td>'ad''verb'</td>
<td>&quot;'ad''verb'&quot;</td>
</tr>
<tr>
<td>&quot;ad&quot;:&quot;verb&quot;</td>
<td>&quot;&quot;ad&quot;:&quot;verb&quot;</td>
</tr>
<tr>
<td>deoxyribonucleotide</td>
<td>&quot;deoxyribonucleotide&quot;</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.

$\text{REVERJw. Format}$

Writes character data in reverse order and preserves blanks.

- **Category:** Character
- **Alignment:** Right

**Syntax**

$\text{REVERJw}$

**Syntax Description**

$w$

specifies the width of the output field.

- **Default:** 1, if $w$ is not specified
- **Range:** 1–32767

**Comparisons**

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

**Example**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>---</td>
</tr>
</tbody>
</table>
$REVERSw. Format

Writes character data in reverse order and left aligns

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

**Syntax**

$REVERSw.$

**Syntax Description**

$w$

- Specifies the width of the output field.
- Default: 1 if $w$ is not specified
- Range: 1–32767

**Comparisons**

The $REVERSw.$ format is similar to the $REVERJw.$ format except that $REVERJw.$ does not left align the result.

**Example**

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

<table>
<thead>
<tr>
<th>Name*</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>------</td>
<td>---------</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ABCD###</th>
<th>DCBA</th>
</tr>
</thead>
<tbody>
<tr>
<td>###ABCD</td>
<td>DCBA</td>
</tr>
</tbody>
</table>

* The character # represents a blank space.
$UPCASEw. Format

Converts character data to uppercase.

Category: Character
Alignment: Left

Syntax

$UPCASEw.

Syntax Description

w

specifies the width of the output field.

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

Range 1–32767

Details

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

Example

put @1 name $upcase9.;

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

$VARYINGw. Format

Writes character data of varying length.

Valid in: in DATA step
Category: Character
Alignment: Left

Syntax

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

Examples

Example 1: Obtaining a Variable Length Directly

An existing data set variable contains the length of a variable. The data values and the results follow the explanation of this SAS statement:

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

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

<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>
</tbody>
</table>
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>Values</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.

Range

1–32767

Comparisons

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

Example

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

<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.

Category: 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

BEST\( w \).

Syntax Description

\( w \)

specifies the width of the output field.
<table>
<thead>
<tr>
<th>Default</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1–32</td>
</tr>
</tbody>
</table>

**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 -0.01 exclusively, use a field width of at least 8.

**Details**

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

**Comparisons**

- The 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 Dw.p format writes numbers with the desired precision and more alignment than the BESTw format.

- The BESTDw.p format is a combination of the BESTw. format and the Dw.p format in that it formats all numeric data, and it does a better job of aligning decimals than the BESTw. format.

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

**Example**

The following statements produce these results:
<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=1257000; put x best6.;</td>
<td>1.26E6</td>
</tr>
<tr>
<td>x=1257000; put x best3.;</td>
<td>1E6</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “BESTDw.p Format” on page 69

---

**BESTDw.p Format**

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

- **Category:** 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**

```
BESTDw.p
```

**Syntax Description**

- **w**
  - specifies the width of the output field.
  - **Default** 12
  - **Range** 1–32

- **p**
  - specifies the precision. This argument is optional.
  - **Default** 3
  - **Range** 0 to w–1
  - **Requirement** must be less than w
  - **Tip** If p is omitted or is specified as 0, then p is set to 3.
Details

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

Example

```
put x bestd14.;
```

<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>123.4500000</td>
</tr>
<tr>
<td>1.2345</td>
<td>1.2345000</td>
</tr>
<tr>
<td>.12345</td>
<td>0.1234500</td>
</tr>
<tr>
<td>1.23456789</td>
<td>1.2345679</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “BESTw. Format” on page 67
- “Dw.p Format” on page 87
**BINARYw. Format**

Converts numeric values to binary representation.

- **Category:** Numeric
- **Alignment:** Left

**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 SBINARYw. format converts character values to binary representation.

**Example**

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>---------</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
</tr>
<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
- **Alignment:** Left
- **Restriction:** UTC time zone offset values are not supported.
Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

\texttt{B8601DAw}.

Syntax Description

\texttt{w}

- specifies the width of the output field.

  Default: \texttt{10}

  Range: \texttt{8-10}

Details

The B8601DA format writes the date value by using the ISO 8601 basic date notation \texttt{yyyymmdd}:

- \texttt{yyyy}: a four-digit year.
- \texttt{mm}: a two-digit month (zero padded) between 01 and 12.
- \texttt{dd}: a two-digit day of the month (zero padded) between 0 and 31.

Example

\texttt{put bda b8601da.;}

<table>
<thead>
<tr>
<th>Value of \texttt{bda}</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>18885</td>
<td>20110915</td>
</tr>
<tr>
<td>18628</td>
<td>20110101</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

\texttt{B8601DNw. Format}

Writes dates from datetime values by using the ISO 8601 basic notation \texttt{yyyymmdd}.

Categories: Date and Time
ISO 8601

Alignment: Left

Restriction: UTC time zone offset values are not supported.
Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

B8601DNw.

Syntax Description

w
  specifies the width of the output field.

Default 10
Range 8–10

Details

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

put bdn b8601dn.;

<table>
<thead>
<tr>
<th>Value of bdn</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1631664000</td>
<td>20110915</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

B8601DTw.d Format

Writes datetime values by using the ISO 8601 basic notation yyyyymmddThhmmss<ffffff>.

Categories: Date and Time
ISO 8601

Alignment: Left

Restriction: UTC time zone offset values are not supported.

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.

Supports: ISO 8601 Element 5.4.1, complete representation

Syntax

**B8601DT** <i>w.d</i>

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 **yyyyymmddTHhmmss<ffffff>**:

- **yyyy** is a four-digit year.
- **mm** is a two-digit month (zero padded) between 01 and 12.
- **dd** is a two-digit day of the month (zero padded) between 01 and 31.
- **hh** is a two-digit hour (zero padded) between 00 and 23.
- **mm** is a two-digit minute (zero padded) between 00 and 59.
- **ss** is a two-digit second (zero padded) between 00 and 59.
- **ffffff** are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```plaintext
put bdt b8601dt.;
```
### Syntax

**B8601DXw.**

### Syntax Description

\[ w \]

- \( w \) 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> ;</td>
<td></td>
</tr>
<tr>
<td>t='01Feb2013T12:34:56'dt ;</td>
<td>20130201T073456-0500</td>
</tr>
<tr>
<td>put t b8601dx.;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>

| options timezone='America/Adak';  |                               |
| data _null_ ;                     | 20130201T073456-0500          |
| t='01Feb2013T12:34:56'dt ;       |                               |
| put t b8601dx.;                   |                               |
| run;                              |                               |

**See Also**

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

**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 *yyyymmddThhmmss*+0000.

**Categories:** Date and Time
ISO 8601

**Alignment:** Left
**Syntax**

B8601DZh.

**Syntax Description**

\( w \)

specifies the width of the output field.

Default 26

Range 16–35

**Details**

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

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

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

\( yyyy \)

is a four-digit year.

\( mm \)

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

\( dd \)

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

\( hh \)

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

\( mm \)

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

\( ss \)

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

\(+0000\)

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

An ISO 8601 time or datetime value that specifies a time zone offset is adjusted by the number of hours and minutes that is specified in the offset. Then, the time zone offset is processed as the time or datetime for the zero meridian (Greenwich, England). The B8601DZ format always writes the datetime value by using the zero meridian offset value of +0000. To write a datetime that uses a time zone offset other than +0000, see “B8601LZw. Format” on page 79.

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

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

**Example**

```
put bdz b8601dz20.;
```

<table>
<thead>
<tr>
<th>Datetime Value</th>
<th>Value of bdz</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>20110915T155300+0500</td>
<td>1631703180</td>
<td>20110915T105300+0000</td>
</tr>
<tr>
<td>20110915T155300Z</td>
<td>1631721180</td>
<td>20110915T105300+0000</td>
</tr>
</tbody>
</table>

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

**See Also**

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations”
on page 14

---

**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 $\text{yyyymmddT}hhmmss+\text{–}hhmm$.

- **Categories:** Date and Time  
  ISO 8601
- **Alignment:** Right  
- **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:

- $\text{yyyymmddT}hhmmss+\text{–}hhmm$
yyyy
  is a four-digit year.

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

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

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

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

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

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

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

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

Example

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

blx='01Feb2013T12:34:56'dt;
p拥护 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 14

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

Supports: ISO 8601 Elements 5.3.3 and 5.3.4.2
Syntax

\texttt{B8601LZ}_w. \\

\textit{Syntax Description}

\texttt{w} \\

specifies the width of the output field.

\textbf{Default} \hspace{1em} 14 \hspace{1em} \textbf{Range} \hspace{1em} 9–20

\textbf{Details}

The \texttt{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 \texttt{hhmmss+|–hhmm}:

\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{+|–hhmm} \\

is an hour and minute signed offset from zero meridian time. Note that the offset must be \texttt{+|–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.

\textbf{Restriction:} The shorter form \texttt{+|–hh} is not supported.

When SAS reads a UTC time by using the \texttt{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 \texttt{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.

\textbf{Example}

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

\texttt{put blz b8601lz.;}

<table>
<thead>
<tr>
<th>Value of \texttt{blz}</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>46380</td>
<td>125300–0500</td>
</tr>
</tbody>
</table>
B8601TMw.d Format

Writes time values by using the ISO 8601 basic notation $hhmmss<ffffff>$.  

**Categories:**  
Date and Time  
ISO 8601

**Alignment:**  
Left

**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

**Syntax**

B8601TMw.d

**Syntax Description**

$\!

w  

specifies the width of the output field.

Default 8

Range 6–15

$d$

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

Default 0

Range 0–6

**Details**

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

$\!

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

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

$ss$  
is a two-digit second (zero padded) between 00 and 59.
are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

put btm b8601tm.;

<table>
<thead>
<tr>
<th>Value of btm</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>57180</td>
<td>155300</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

## B8601TXw. Format

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 $hhmsss|+|-hhmm$.

**Categories:** Date and Time

**ISO 8601**

**Alignment:** Right

**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:

- $hhmsss|+|-hhmm$

$hh$

is a two-digit hour (zero padded) between 00 and 23.
**Statement** | **Result**
---|---
data _null_;  
t='12:34:56't;  
put t b8601tx.;  
run;  
| 073456-0500

options timezone='America/Adak';  
data _null_;  
t='12:34:56't;  
put t b8601tx.;  
run;  
| 023456-1000

**See Also**

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

---

**B8601TZw. Format**

Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 basic time notation \( hhmmss [+\text{–}] hhmm \).

**Categories:** Date and Time  
ISO 8601

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

Syntax

B8601TZw.

Syntax Description

w
specifies the width of the output field.

Default 14
Range 9–20

Details

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

- \( hhmmss+|–hhmm \)
  
  Note: Use this form when \( w \) is large enough to support this time notation.

- \( hhmmssZ \)
  
  Note: Use this form when \( w \) is not large enough to support the \( +|–hhmm \) time zone notation.

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

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

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

\( +|–hh:mm \)

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

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

Restriction: The shorter form \( +|–hh \) is not supported.

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

When SAS reads a UTC time by using the B8601TZ informat, and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 000000 and 240000. If the B8601TZ format attempts to format a time outside of this time range, the time is formatted with asterisks to indicate that the value is out of range.
Comparisons
For time values between 000000 and 240000, the B8601TZ format adjusts the time value to be the time at the zero meridian and writes the time value in the international standard extended time notation. The B8601LZ format makes no adjustment to the time and writes time values in the international standard extended time notation, using a UTC time zone offset for the local SAS session.

Example
put btz b8601tz.;

<table>
<thead>
<tr>
<th>Values for btz</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>73441</td>
<td>202401+0000</td>
</tr>
</tbody>
</table>

See Also
“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

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

**Category:** Numeric  
**Alignment:** Right  
**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**
COMMA_w.d

**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>6</td>
<td>1–32</td>
<td>Make w wide enough to write the numeric values, the commas, and the optional decimal point.</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.
Details

The COMMAw.d format writes numeric values with a comma that separates every three digits and a period 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.
- The COMMAw.d format is similar to the DOLLARw.d format except that the COMMAw.d format does not print a leading dollar sign.

Example

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

<table>
<thead>
<tr>
<th>Value of sales</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:

- “COMMAXw.d Format” on page 86
- “DOLLARw.d Format” on page 98
Syntax
COMMAX \( w.d \)

**Syntax Description**

\( w \)

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

- Default: 6
- Range: 1–32
- Tip: Make \( w \) wide enough to write the numeric values, the commas, and the optional decimal point.

\( d \)

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

- Range: 0–31
- Requirement: must be less than \( w \)

**Details**

The COMMAX \( w.d \) format writes numeric values with a period that separates every three digits and with a comma that separates the decimal fraction.

**Comparisons**

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

**Example**

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

<table>
<thead>
<tr>
<th>Value of sales</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.

**Category:** Numeric
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**

\[D_{w,p}\]

**Syntax Description**

\(w\)

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

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

\(p\)

specifies the precision. This argument is optional.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0–9</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 \(D_{w,p}\) format.

**Details**

The \(D_{w,p}\) format writes numbers so that the decimal point aligns in groups of values with similar magnitude. Larger values of \(p\) print the data values with more precision and potentially more shifts in the decimal point alignment. Smaller values of \(p\) print the data values with less precision and a greater chance of decimal point alignment.

**Comparisons**

- The BEST\(w\) format writes as many significant digits as possible in the output field, but if the numbers vary in magnitude, the decimal points do not line up.
- \(D_{w,p}\) writes numbers with the desired precision and more alignment than the BEST\(w\) format.
- The BESTD\(w,p\) format is a combination of the BEST\(w\) format and the \(D_{w,p}\) format in that it formats all numeric data, and it does a better job of aligning decimals than the BEST\(w\) format.
- The \(w.d\) format aligns decimal points, if possible, but it does not necessarily show the same precision for all numbers.
Example

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

<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>123.45</td>
<td>123.4500</td>
</tr>
<tr>
<td>12.345</td>
<td>12.3450</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,p Format” on page 69

---

**DATEw. Format**

Writes date values in the form *ddmmyy*, *ddmmyyyy*, or *dd-mm-yyyy*.

**Category:** 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 \textit{ddmmyy}, \textit{ddmmyyyy}, or \textit{dd-mm-yyy}, where

\begin{itemize}
  \item \textit{dd} is an integer that represents the day of the month.
  \item \textit{mmm} is the first three letters of the month name.
  \item \textit{yy} or \textit{yyyy} is a two-digit or four-digit integer that represents the year.
\end{itemize}

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{put day date5.;}</td>
<td>\texttt{16MAR}</td>
</tr>
<tr>
<td>\texttt{put day date6.;}</td>
<td>\texttt{16MAR}</td>
</tr>
<tr>
<td>\texttt{put day date7.;}</td>
<td>\texttt{16MAR12}</td>
</tr>
<tr>
<td>\texttt{put day date8.;}</td>
<td>\texttt{16MAR12}</td>
</tr>
<tr>
<td>\texttt{put day date9.;}</td>
<td>\texttt{16MAR2012}</td>
</tr>
<tr>
<td>\texttt{put day date11.;}</td>
<td>\texttt{16-MAR-2012}</td>
</tr>
</tbody>
</table>

See Also

Functions:
- “DATE Function” in \textit{SAS Functions and CALL Routines: Reference}

Informats:
- “DATEw. Informat” on page 300

\section*{DATEAMPMw.d Format}

Writes datetime values in the form \textit{ddmmyy:hh:mm:ss.ss} with AM or PM.

\begin{itemize}
  \item \textbf{Category:} Date and Time
  \item \textbf{Alignment:} Right
\end{itemize}
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

DATEAMPMw.d

Syntax Description

w
specifies the width of the output field.

Default 19

Range 7–40

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

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

Range 0–39

Requirement must be less than w

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

Details

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

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

mmm
is the first three letters of the month name.

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

hh
is an integer that represents the hour.

mm
is an integer that represents the minutes.

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

Comparisons

The DATEAMPMw.d format is similar to the DATETIMEw.d format except that DATEAMPMw.d prints AM or PM at the end of the time.
Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put event dateampm.;</td>
<td>20APR12:11:01:34 AM</td>
</tr>
<tr>
<td>put event dateampm7.;</td>
<td>20APR12</td>
</tr>
<tr>
<td>put event dateampm10.;</td>
<td>20APR:11</td>
</tr>
<tr>
<td>put event dateampm13.;</td>
<td>20APR12:11 AM</td>
</tr>
<tr>
<td>put event dateampm22.2;</td>
<td>20APR12:11:01:34.00 AM</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DATETIMEw.d Format” on page 92

DATETIMEw.d Format

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

**Category:** Date and Time

**Alignment:** Right

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

**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.

**Default** 16

**Range** 7–40
Tip SAS requires a minimum $w$ value of 16 to write a SAS datetime value with the date, hour, and seconds. Add an additional two places to $w$ and a value to $d$ to return values with optional decimal fractions of seconds.

$d$

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

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

Details

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

$dd$

is an integer that represents the day of the month.

$mmm$

is the first three letters of the month name.

$yy$

is a two-digit integer that represents the year.

$hh$

is an integer that represents the hour in 24–hour clock time.

$mm$

is an integer that represents the minutes.

$ss.ss$

is the number of seconds to two decimal places.

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put event datetime.;</td>
<td>10NOV12:03:49:19</td>
</tr>
<tr>
<td>put event datetime7.;</td>
<td>10NOV12</td>
</tr>
<tr>
<td>put event datetime12.;</td>
<td>10NOV12:03</td>
</tr>
<tr>
<td>put event datetime18.;</td>
<td>10NOV12:03:49:19</td>
</tr>
<tr>
<td>put event datetime18.1;</td>
<td>10NOV12:03:49:19.0</td>
</tr>
<tr>
<td>put event datetime19.;</td>
<td>10NOV2012:03:49:19</td>
</tr>
</tbody>
</table>
### DAYw. Format

Writes date values as the day of the month.

**Category:** Date and Time  
**Alignment:** Right

#### Syntax

\[
\text{DAYw.}
\]

#### Syntax Description

\[
w
\]

Specifies the width of the output field.

- **Default:** 2
- **Range:** 2–32

#### Example

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

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

\begin{itemize}
  \item \textbf{Category:} Date and Time
  \item \textbf{Alignment:} Right
\end{itemize}

\textbf{Syntax}

\texttt{DDMMYYw.}

\textbf{Syntax Description}

\textit{w}

\begin{itemize}
  \item \textbf{specifies the width of the output field.}
  \item \textbf{Default} 8
  \item \textbf{Range} 2–10
  \item \textbf{Interaction} When \textit{w} has a value of from 2 to 5, the date appears with as much of the day and the month as possible. When \textit{w} is 7, the date appears as a two-digit year without slashes.
\end{itemize}

\textbf{Details}

The \texttt{DDMMYYw.} format writes SAS date values in the form \textit{ddmm<yy>yy} or \textit{dd/mm/<yy>yy}:

\textit{dd}

\begin{itemize}
  \item is an integer that represents the day of the month.
\end{itemize}

\textit{/}

\begin{itemize}
  \item is the separator.
\end{itemize}

\textit{mm}

\begin{itemize}
  \item is an integer that represents the month.
\end{itemize}

\textit{<yy>yy}

\begin{itemize}
  \item is a two-digit or four-digit integer that represents the year.
\end{itemize}

\textbf{Example}

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

\begin{verbatim}
put date day2.; 14
\end{verbatim}
SAS Statement | Result  
---+----
put date ddmmyy5.; 24/12
put date ddmmyy6.; 241212
put date ddmmyy7.; 241212
put date ddmmyy8.; 24/12/12
put date ddmmyy10.; 24/12/2012

See Also

Formats:
- “DATEw. Format” on page 89
- “DDMMYYxw. Format” on page 96
- “MMDDYYw. Format” on page 137
- “YYMMDDw. Format” on page 209

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

Informats:
- “DATEw. Informat” on page 300
- “DDMMYYw. Informat” on page 303
- “MMDDYYw. Informat” on page 327
- “YYMMDDw. Informat” on page 385

**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, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.

**Category:** Date and Time  
**Alignment:** Right

**Syntax**  
`DDMMYYxw.`
Syntax Description

$x$

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

- B: separates with a blank
- C: separates with a colon
- D: separates with a hyphen
- N: indicates no separator
- P: separates with a period
- S: separates with a slash.

$w$

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>2–10</td>
</tr>
</tbody>
</table>

Interactions

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

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

Details

The DDMMYY$wxw$. format writes SAS date values in the form $ddmm<y>y$ or $dxmxx<y>y$:

- $dd$ is an integer that represents the day of the month.
- $x$ is a specified separator.
- $mm$ is an integer that represents the month.
- $<y>y$ is a two-digit or four-digit integer that represents the year.

Example

The following examples use the input value of 19137, which is the SAS date value that corresponds to May 24, 2012.
SAS Statement | Result
---+----|---+
put date ddmmyyc5.; | 24:05
put date ddmmyyd8.; | 24-05-12
put date ddmmyyp10.; | 24.05.2012
put date ddmmyyn8.; | 24052012

See Also

Formats:
- “DATEw. Format” on page 89
- “DDMMYYw. Format” on page 95
- “MMDDYYxw. Format” on page 139
- “YYMMDDxw. Format” on page 210

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 303

DOLLARw.d Format

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

Category: 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

DOLLARw.d
Syntax Description

\( w \)

specifies the width of the output field.

Default: 6

Range: 2–32

\( d \)

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

Range: 0–31

Requirement: must be less than \( w \)

Details

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

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

Comparisons

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

Example

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

<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:

- “COMMA\( w \).\( d \) Format” on page 85
- “DOLLARX\( w \).\( d \) Format” on page 100
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.

**Category:** 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

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

### Syntax Description

\( w \)

specifies the width of the output field.

- **Default:** 6
- **Range:** 2–32

\( d \)

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

- **Default:** 0
- **Range:** 0–31
- **Requirement:** must be less than \( w \)

### Details

The DOLLARXw.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 DOLLARXw.d always produces one of these codes. If you need another monetary character, define your own format with the FORMAT procedure. See For details, see “FORMAT” in Base SAS Procedures Guide.

### Comparisons

- The DOLLARXw.d format is similar to the DOLLARw.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 DOLLARXw.d format is the same as the COMMAXw.d format except that the COMMAw.d format does not write a leading dollar sign.

**Example**

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

<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 86
- “DOLLARw.d Format” on page 98

---

**DOWNAMEw. Format**

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

**Category:** Date and Time

**Alignment:** Right

**Syntax**

`DOWNAMEw.`

**Syntax Description**

`w`

specifies the width of the output field.

- **Default**: 9
- **Range**: 1–32
- **Tip**: If you omit `w`, SAS prints the entire name of the day.

**Details**

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

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

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

**See Also**

**Formats:**
- “WEEKDAYw. Format” on page 196

**DTDATEw. Format**

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

**Category:** Date and Time

**Alignment:** Right

**Syntax**

`DTDATEw.`

**Syntax Description**

`w`

specifies the width of the output field.

- **Default:** 7
- **Range:** 5–9
- **Tip:** Use a width of 9 to print a four-digit year.

**Details**

The `DTDATEw.` format writes SAS date values in the form `ddmmmyy` or `ddmmmyyyy`, where

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

- `mmm` are the first three letters of the month name.

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

The 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

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put trip_date=dtdate.;</td>
<td>16APR12</td>
</tr>
<tr>
<td>put trip_date=dtdate9.;</td>
<td>16APR2012</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 89

DTMONYYw. Format

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

- **Category:** 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 mmyyyy, where

- mmm
  - is the first three letters of the month name.
yy or yyyy

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

Comparisons

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

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date dtmonyy.;</td>
<td>OCT12</td>
</tr>
<tr>
<td>put date dtmonyy5.;</td>
<td>OCT12</td>
</tr>
<tr>
<td>put date dtmonyy6.;</td>
<td>OCT12</td>
</tr>
<tr>
<td>put date dtmonyy7.;</td>
<td>OCT2012</td>
</tr>
</tbody>
</table>

See Also

Formats:

• “DATETIMEw.d Format” on page 92
• “MONYYw. Format” on page 147

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).

Category: Date and Time
Alignment: Right

Syntax

DTWKDATXw.

Syntax Description

w

specifies the width of the output field.
Details

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

- **day-of-week** is either the first three letters of the day name or the entire day name.
- **dd** is an integer that represents the day of the month.
- **month-name** is either the first three letters of the month name or the entire month name.
- **yy** or **yyyy** is a two-digit or four-digit integer that represents the year.

Comparisons

The 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

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date dtwkdatx.;</td>
<td>Tuesday, 16 October 2012</td>
</tr>
<tr>
<td>put date dtwkdatx1.;</td>
<td>Tue</td>
</tr>
<tr>
<td>put date dtwkdatx8.;</td>
<td>Tue</td>
</tr>
<tr>
<td>put date dtwkdatx25.;</td>
<td>Tuesday, 16 Oct 2012</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEVALUEw.d Format” on page 92
- “WEEKDATXw. Format” on page 195
DTYEARw. Format

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

**Category:** Date and Time  
**Alignment:** Right

**Syntax**

```
DTYEARw.
```

**Syntax Description**

`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**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date dtyear.;</td>
<td>2012</td>
</tr>
<tr>
<td>put date dtyear2.;</td>
<td>12</td>
</tr>
<tr>
<td>put date dtyear3.;</td>
<td>12</td>
</tr>
<tr>
<td>put date year4.;</td>
<td>2012</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “DATETIMEw.d Format” on page 92
- “YEARw. Format” on page 206
### DTYYQCw. Format

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

- **Category:** Date and Time
- **Alignment:** Right

#### Syntax

**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

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date dtyyqc.;</td>
<td>12:4</td>
</tr>
<tr>
<td>put date dtyyqc4.;</td>
<td>12:4</td>
</tr>
<tr>
<td>put date dtyyqc5.;</td>
<td>12:4</td>
</tr>
<tr>
<td>put date dtyyqc6.;</td>
<td>2012:4</td>
</tr>
</tbody>
</table>

#### See Also

**Formats:**

- “DATETIMEw.d Format” on page 92
Ew. Format

Writes numeric values in scientific notation.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment:</td>
<td>Right</td>
</tr>
</tbody>
</table>

Syntax

\[ Ew. \]

Syntax Description

\( w \)

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

<table>
<thead>
<tr>
<th>Default</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>7–32</td>
</tr>
</tbody>
</table>

Details

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

Example

put @1 x e10.;

<table>
<thead>
<tr>
<th>Value of ( x )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1.000E+00</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: | 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

put eda e8601da.;

<table>
<thead>
<tr>
<th>Value for eda</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>19251</td>
<td>2012-09-15</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

E8601DNw. Format

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

Categories: Date and Time
ISO 8601

Alignment: Left

Alias: IS8601DN

Restriction: UTC time zone offset values are not supported.
Supports: ISO 8601 Element 5.2.1.1, complete representation

Syntax

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.

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

Example

put edn e8601dn.;

<table>
<thead>
<tr>
<th>Value for edn</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1663308532</td>
<td>2012-09-15</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

E8601DTw.d Format

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

Categories: 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

E8601DT\textit{w.d}

Syntax Description

\textit{w}

specifies the width of the input field.

Default 19

Range 19–26

\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

Details

The E8601DT format writes datetime values by using the ISO 8601 extended datetime notation \texttt{yyyy-mm-ddThh:mm:ss.ffffff}:

\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.

\textit{ffffff}

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

Example

\texttt{put edt e8601dt25.3;}
E8601DXw. Format

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-ddThh:mm:ss+hh:mm.

**Syntax**

E8601DXw:

**Syntax Description**

w

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>26</td>
<td>20–35</td>
</tr>
</tbody>
</table>

**Details**

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

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

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

+−hh:mm
is an hour and minute signed offset from zero meridian time. The offset must be +−hh:mm (that is, + or – and four characters).

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

Restriction: The shorter form +−hh is not supported.

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

<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='01Feb2013T12:34:56'dt ;</td>
<td>2013-02-01T07:34:56-05:00</td>
</tr>
<tr>
<td>put t e8601dx.;</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
<tr>
<td>options timezone='America/Adak';</td>
<td>2013-02-01T02:34:56-10:00</td>
</tr>
<tr>
<td>data <em>null</em> ;</td>
<td></td>
</tr>
<tr>
<td>t='01Feb2013T12:34:56'dt ;</td>
<td></td>
</tr>
<tr>
<td>put t e8601dx.;</td>
<td></td>
</tr>
<tr>
<td>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 14

E8601DZw. Format
Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss±0:00.

Categories: Date and Time
ISO 8601

Alignment: Left

Alias: IS8601DZw.

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.1, complete representation
Syntax

\texttt{E8601DZw}.

**Syntax Description**

\texttt{w}

specifies the width of the output field.

**Default**

26

**Range**

20–35

**Details**

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

- \texttt{yyyy-mm-ddThh:mm:ss+00:00}

  \textit{Note:} Use this form when \texttt{w} is large enough to support this time zone notation.

- \texttt{yyyy-mm-ddThh:mm:ssZ}

  \textit{Note:} Use this form when \texttt{w} is not large enough to support the +00:00 time zone notation.

\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 24.

\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{+00:00}

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

An ISO 8601 time or datetime value that specifies a time zone offset is adjusted by the number of hours and minutes that is specified in the offset and processed as the time or datetime for the zero meridian (Greenwich, England). The E8601DZ format always writes the datetime value by using the zero meridian offset value of +00:00. To write a datetime that uses the UTC offset other than +00:00, see “E8601LZw. Format” on page 116.

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

\texttt{Z}

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

put edz e8601dz.;

<table>
<thead>
<tr>
<th>Value of edz</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1663332780</td>
<td>2012-09-15T12:53:00+00:00</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

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:** 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 the 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.

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

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

$\pm hh:mm$ is an hour and minute signed offset from zero meridian time. The offset must be $+\vert- hh:mm$ (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 $\pm hh$ is not supported.

**Example**

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

``` SAS
blx='01Feb2013T12:34:56'dt;
put blx e8601lx.;
```

<table>
<thead>
<tr>
<th>Value of blx</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1675341296</td>
<td>2013-02-01T12:34:56-05:00</td>
</tr>
</tbody>
</table>

**See Also**

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

**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 $hh:mm:ss \pm hh:mm$.

**Categories:** 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 the following ISO 8601 extended time notations:

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

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

\( hh \)

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

\( mm \)

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

\( ss \)

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

\( +|–hh:mm \)

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

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

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

\( Z \)

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

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

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

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

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

```
put elz e8601lz.;
```

<table>
<thead>
<tr>
<th>Value of elz</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>46380</td>
<td>12:53:00-5:00</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

E8601TMw.d Format

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

| Categories:      | Date and Time
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO 8601</td>
<td></td>
</tr>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
<tr>
<td>Alias:</td>
<td>IS8601TMw.d</td>
</tr>
<tr>
<td>Restriction:</td>
<td>UTC time zone offset values are not supported.</td>
</tr>
<tr>
<td>Interaction:</td>
<td>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.</td>
</tr>
<tr>
<td>Supports:</td>
<td>ISO 8601 Element 5.3.1.1, complete representation, and 5.3.1.3, representation of decimal fractions</td>
</tr>
</tbody>
</table>

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.

**Default**: 0

**Range**: 0–6
Details

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

- `hh` is a two-digit hour (zero padded) between 00 and 23.
- `mm` is a two-digit minute (zero padded) between 00 and 59.
- `ss` is a two-digit second (zero padded) between 00 and 59.
- `ffffff` are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Example

```sas
put etm e8601tm.;
```

<table>
<thead>
<tr>
<th>Value of <code>etm</code></th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>57180</td>
<td>15:53:00</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14

---

E8601TXw. Format

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`.

**Categories:** 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 the following 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> ; t='12:34:56't; put t e8601tx.; run;</td>
<td>07:34:56-05:00</td>
</tr>
<tr>
<td>options timezone='America/Adak'; data <em>null</em> ; t='12:34:56't; put t e8601tx.; run;</td>
<td>02:34:56-10:00</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14
**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:** Date and Time
- ISO 8601

**Alignment:** Left

**Alias:** IS8601TZw.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.

**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

---

**Details**

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

- `hh:mm:ss<fff>+-hh:mm`
  
  **Note:** Use this form when `w` is large enough to support this time zone notation.

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

`hh`
- is a two-digit hour (zero padded) between 00 and 23.
**FLOAT**<sup>w.d</sup> Format

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

---

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

**Example**

```sas
put etz e8601tz.;
```

<table>
<thead>
<tr>
<th>Value of etz</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>17024</td>
<td>04:43:44+00:00</td>
</tr>
<tr>
<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 14
Syntax
FLOAT\(w.d\)

Syntax Description
\(w\)

specifies the width of the output field.

Requirement  width must be 4

\(d\)

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

Default 0

Range 0-31

Details

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

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

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

Comparisons

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

<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>Language</td>
<td>Float Notation</td>
</tr>
<tr>
<td>----------</td>
<td>----------------</td>
</tr>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
</tr>
</tbody>
</table>

**Example**

```plaintext
put x float4.;
```

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

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

---

**FRACTw. Format**

Converts numeric values to fractions.

- **Category:** 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, that is, 1/2 instead of 50/100.

**Example**

```plaintext
put x fract8.;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---</td>
<td>+-----1</td>
</tr>
</tbody>
</table>
### HEXw. Format

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

**Category:** Numeric  
**Alignment:** Left  
**See:**  
“HEXw. Format: Windows” in SAS Companion for Windows  
“HEXw. Format: UNIX” in SAS Companion for UNIX Environments

### 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

```
put @8 x hex$.;
```

<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>00000002</td>
</tr>
<tr>
<td>-150</td>
<td>FFFFFFF6A</td>
</tr>
</tbody>
</table>

**HHMMw.d Format**

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

**Category:** 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 \textit{hh:mm}:

\textit{hh}

is an integer.

\textit{Note:} If \textit{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.

\textit{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

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

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

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

See Also

Formats:

- “HOURw.d Format” on page 128
- “MMSSw.d Format” on page 141
- “TIMEw.d Format” on page 184
- “TODw.d Format” on page 187

Functions:

- “HMS Function” in \textit{SAS Functions and CALL Routines: Reference}
Informats:
- “TIMEw. Informat” on page 369

**HOURw.d Format**

Writes time values as hours and decimal fractions of hours.

**Syntax**

\[ \text{HOURw.d} \]

**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**

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

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

**Category:** Numeric  
**Alignment:** Left

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

#### Syntax

**IBw.d**

**Syntax Description**

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

**Default**  
4
Range 1–8

$d$

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

Default 0

Range 0–10

Details

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

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 8.

Comparisons

The IB$w.d$ and PIB$w.d$ formats are used to write native format integers. (Native format enables you to read and write values created in the same operating environment.) The IB$Rw.d$ and PIB$Rw.d$ 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 or Little Endian Platforms” on page 8.

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

Example

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

<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>1</td>
<td>00000001</td>
<td>80000000</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:

• “IB$Rw.d$ Format” on page 131
IBRw.d Format

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

Category: Numeric
Alignment: Left

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 \(10^d\). 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 8.

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 or Little Endian Platforms” on page 8.
To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 9.

Example

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

<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 129

### IEEEw.d Format

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

**Category:** Numeric  
**Alignment:** Left  
**CAUTION:** Large floating-point values and floating-point values that require precision might not be identical to the original SAS value when they are written to an IBM mainframe 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.

- **Default**: 8  
- **Range**: 3–8

**Tip**

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

\[ d \]

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

**Default**

0

**Range**

0–10

**Details**

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

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

**Example**

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

<table>
<thead>
<tr>
<th>Value of (x)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3F800000</td>
</tr>
<tr>
<td></td>
<td>3FF00000000</td>
</tr>
</tbody>
</table>

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

**JULDAYw. Format**

Writes date values as the Julian day of the year.

**Category:** Date and Time

**Alignment:** Right

**Syntax**

JULDAY\(w\).
Syntax Description

\( w \)

specifies the width of the output field.

Default: 3

Range: 3–32

Details

The JULDAY\( w \). format writes SAS date values in the form \( ddd \), where

\( ddd \)

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

Example

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

<table>
<thead>
<tr>
<th>Input Value</th>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>18993</td>
<td>put date julday3.;</td>
<td>1</td>
</tr>
<tr>
<td>19068</td>
<td>put date julday3.;</td>
<td>76</td>
</tr>
</tbody>
</table>

JULIAN\( w \). Format

Writes date values as Julian dates in the form \( yyddd \) or \( yyyyddd \).

Category: Date and Time

Alignment: Left

Syntax

JULIAN\( w \).

Syntax Description

\( w \)

specifies the width of the output field.

Default: 5

Range: 5–7
Tip

If \( w \) is 5, the JULIAN\( w \) format writes the date with a two-digit year. If \( w \) is 7, the JULIAN\( w \) format writes the date with a four-digit year.

Details

The JULIAN\( w \) format writes SAS date values in the form \( yyddd \) or \( yyyyddd \):

- \( yy \) or \( yyyy \)
  - is a two-digit or four-digit integer that represents the year.
- \( ddd \)
  - is the number of the day, 1–365 (or 1–366 for leap years), in that year.

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date julian5.;</td>
<td>12122</td>
</tr>
<tr>
<td>put date julian7.;</td>
<td>2012122</td>
</tr>
</tbody>
</table>

See Also

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

Informats:
- “JULIANw. Informat” on page 324

MDYAMP\( w.d \) Format

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

Category: Date and Time

Alignment: Right

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: The default time period is AM.
Syntax

MDYAMPM\textit{w}.

Syntax Description

\textit{w}

specifies the width of the output field.

Default 19

Range 8–40

Details

The MDYAMPM\textit{w.d} format writes SAS datetime values in the following form:

\textit{mm/dd/yy<yy> hh:mm<AM | PM>}

\textit{mm}

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

\textit{dd}

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

\textit{yy} or \textit{yyyy}

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

\textit{hh}

is an integer between 00 and 23 that represents hours.

\textit{mm}

is an integer between 00 and 59 that represents minutes.

\textit{AM | PM}

specifies either the time period 00:01–12:00 noon (PM) or the time period

12:01–12:00 midnight (AM). The default is AM.

date and time separator characters

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

Comparisons

The MDYAMPM\textit{w} format writes datetime values with separators in the form \textit{mm/dd/}

\textit{yy<yy> hh:mm AM | PM}, and requires a space between the date and the time.

The DATETIME\textit{w.d} format writes datetime values with separators in the form

\textit{ddmmmyy<yy>: hh:mm:ss.ss}.

Example

This example uses the input value of 1663343580, which is the SAS datetime value that

corresponds to 3:53:00 PM on September 15, 2012.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put dt mdyampm25.</td>
<td>9/15/2012 3:53 PM</td>
</tr>
</tbody>
</table>
See Also

Formats:

• “DATETIMEw.d Format” on page 92

Informats:

• “MDYAMPMw.d Informat” on page 325

**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 2 or 4 digits.

**Category:** Date and Time  
**Alignment:** Right

**Syntax**

`MMDDYYw.`

**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 of from 2 to 5, the date appears with as much of the month and the day as possible. When <code>w</code> is 7, the date appears as a two-digit year without slashes.</td>
</tr>
</tbody>
</table>

**Details**

The MMDDYYw. format writes SAS date values in one of the following forms:

`mmdd<yy>yy`

`mm/dd/<yy>yy`

where

`mm`

is an integer that represents the month.

`/`

is the separator.

`dd`

is an integer that represents the day of the month.

`<yy>yy`

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

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put day mmddyy2.;</td>
<td>10</td>
</tr>
<tr>
<td>put day mmddyy3.;</td>
<td>10</td>
</tr>
<tr>
<td>put day mmddyy4.;</td>
<td>1025</td>
</tr>
<tr>
<td>put day mmddyy5.;</td>
<td>10/25</td>
</tr>
<tr>
<td>put day mmddyy6.;</td>
<td>102512</td>
</tr>
<tr>
<td>put day mmddyy7.;</td>
<td>102512</td>
</tr>
<tr>
<td>put day mmddyy8.;</td>
<td>10/25/12</td>
</tr>
<tr>
<td>put day mmddyy10.;</td>
<td>10/25/2012</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 89
- “DDMMYYYw. Format” on page 95
- “MMDDYYYxw. Format” on page 139
- “YYMMDDDw. Format” on page 209

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 300
- “DDMMYYw. Informat” on page 303
- “YYMMDDDw. Informat” on page 385
MMDDYYxw. Format

Writes date values in the form \textit{mmdd\textless yy\textgreater yy} or \textit{mm-dd\textless yy\textgreater 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 2 or 4 digits.

\begin{itemize}
  \item \textbf{Category}: Date and Time
  \item \textbf{Alignment}: Right
\end{itemize}

\section*{Syntax}

\texttt{MMDDYYxw}.

\section*{Syntax Description}

\textit{x}

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

- \textit{B} separates with a blank.
- \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 slash.

\textit{w}

specifies the width of the output field.

\begin{itemize}
  \item \textbf{Default} 8
  \item \textbf{Range} 2–10
\end{itemize}

\section*{Interactions}

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

When \textit{x} has a value of N, the width range changes to 2–8.

\section*{Details}

The \texttt{MMDDYYxw} format writes SAS date values in one of the following forms:

\textit{mmdd\textless yy\textgreater yy}
where

\( \text{mm} \)

is an integer that represents the month.

\( x \)

is a specified separator.

\( \text{dd} \)

is an integer that represents the day of the month.

\( \text{<yy>yy} \)

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

### Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put day mmddyc5.;</td>
<td>05:14</td>
</tr>
<tr>
<td>put day mmddyyd8.;</td>
<td>05-14-12</td>
</tr>
<tr>
<td>put day mmddyp10.;</td>
<td>05.14.2012</td>
</tr>
<tr>
<td>put day mmddyn8.;</td>
<td>05142012</td>
</tr>
</tbody>
</table>

### See Also

**Formats:**
- “DATEw. Format” on page 89
- “DDMMYYxw. Format” on page 96
- “MMDDYYw. Format” on page 137
- “YYMMDDxw. Format” on page 210

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

**Informats:**
- “MMDDYYw. Informat” on page 327
**MMSSw.d Format**

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

**Category:** 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

`MMSSw.d`

### Syntax Description

**w**  
specifies the width of the output field.  
- Default: 5  
- Range: 2–20  
- Tip: Set w to a minimum of 5 to write a value that represents minutes and seconds.

**d**  
specifies the number of digits to the right of the decimal point in the seconds value. Therefore, the SAS time value includes fractional seconds. This argument is optional.  
- Range: 0–19  
- Restriction: must be less than w

### Details

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

### Example

The example uses the input value of 4530.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put time mmss.;</td>
<td>75:30</td>
</tr>
</tbody>
</table>
MMYYw. Format

W riting date values in the form \( mmM\textless yy\textgreater yy \), where \( M \) is the separator and the year appears as either 2 or 4 digits.

**Category:** Date and Time

**Alignment:** Right

**Syntax**

\[ \text{MMYYw.} \]

**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 \( mmM\textless yy\textgreater yy \), where

\( mm \)

is an integer that represents the month.

\( M \)

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

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date mmyy5.;</td>
<td>10M12</td>
</tr>
<tr>
<td>put date mmyy6.;</td>
<td>10M12</td>
</tr>
<tr>
<td>put date mmyy.;</td>
<td>10M2012</td>
</tr>
<tr>
<td>put date mmyy7.;</td>
<td>10M2012</td>
</tr>
<tr>
<td>put date mmyy10.;</td>
<td>10M2012</td>
</tr>
</tbody>
</table>

See Also

Formats:

• “MMYYxw. Format” on page 143
• “YYMMw. Format” on page 207

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

Category: Date and Time
Alignment: Right

Syntax

\(\text{MMYYxw}\).

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.

<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 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 MMYYwxw. format writes SAS date values in one of the following forms:

\[ mm<yy>yy \]
\[ mm<yyxxyy \]
\[ mmx<yy>yy \]

where

\[ mm \]
  is an integer that represents the month.

\[ x \]
  is a specified separator.

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

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date mmyyc5.;</td>
<td>05:12</td>
</tr>
</tbody>
</table>
SAS Statement | Result
---|---
put date mmyyd.; | 05-2012
put date mmyyn4.; | 0512
put date mmyyp8.; | 05.2012
put date mmyys10.; | 05/2012

See Also

Formats:
- “MMYYw. Format” on page 142
- “YYMMxw. Format” on page 212

**MONNAMEw. Format**

Writes date values as the name of the month.

- Category: 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**

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

Writes date values as the month of the year.

**Category:** Date and Time

**Alignment:** Right

**Syntax**

MONTH\(w\).

**Syntax Description**

\(w\)

specifies the width of the output field.

**Default** 2

**Range** 1–32

**Tip** Use MONTH1. to obtain a hexadecimal value.

**Details**

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

**Example**

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

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

**Category:** Date and Time  
**Alignment:** Right

#### Syntax

\[ \text{MONYY}w. \]

#### 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`, where

- `mmm` is the first three letters of the month name.  
- `yy` or `yyyy` is a two-digit or four-digit integer that represents the year.

#### Comparisons

The 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

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

Writes negative numeric values in parentheses.

**Category:** 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**

```
NEGPARNW.d
```

**Syntax Description**

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

  **Default:** 6  
  **Range:** 1–32

---

### See Also

**Formats:**  
- “DDMMYYw. Format” on page 95  
- “DTMONNYw. Format” on page 103  
- “MMDDYYw. Format” on page 137  
- “YYMMDDw. Format” on page 209

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

**Informat:**  
- “MONYYw. Informat” on page 328

---

### SAS Statement Result

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date monyy5.;</td>
<td>MAY12</td>
</tr>
<tr>
<td>put date monyy7.;</td>
<td>MAY2012</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.

Default 0

Range 0–31

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

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

Example

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

<table>
<thead>
<tr>
<th>Value of sales</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>

NUMX\( w.d \) Format

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

- **Category:** 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
NUMX\(w.d\)

Syntax Description

\(w\)

specifies the width of the output field.

Default 12

Range 1–32

\(d\)

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

Default 0

Range 0–31

Details

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

Comparisons

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

Example: Examples

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

<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 192

Informs:

- “NUMXw.d Informat” on page 331
**OCTALw. Format**

Converts numeric values to octal representation.

- **Category:** Numeric
- **Alignment:** Left

**Syntax**

```octal
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 $OCTALw.$ format converts character values to octal representation.

**Example**

```plaintext
put x octal6.;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>007010</td>
</tr>
</tbody>
</table>

**PDw.d Format**

Writes data in packed decimal format.

- **Category:** Numeric
- **Alignment:** Left
- **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: “PDw.d Format: UNIX” in SAS Companion for UNIX Environments
“PDw.d Format: Windows” in SAS Companion for Windows
“PDw.d Format: z/OS” in SAS Companion for z/OS

Syntax
PDw.d

Syntax Description

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

Default 1
Range 1–16

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

Default 0
Range 0–31

Details

Different operating environments store packed decimal values in different ways. However, the PDw.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 PDw.d format writes missing numerical data as −0. When the PDw.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
y=put(x,pd4.);
put y $hex8.;

| Value of x | Result  *
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>00000128</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.

**PDJULGw. Format**

Writes packed Julian date values in the hexadecimal format `yyyydddF` for IBM.

**Category:** Date and Time

**Syntax**

`PDJULGw.`

**Syntax Description**

`w`

specifies the width of the output field.

Default 4

Range 3-16

**Details**

The PDJULGw. format writes SAS date values in the form `yyyydddF`:

- `yyyy` is the two-byte representation of the four-digit Gregorian year.
- `ddd` is the one-and-a-half byte representation of the three-digit integer that corresponds to the Julian day of the year, 1–365 (or 1–366 for leap years).
- `F` is the half byte that contains all binary 1s, which assigns the value as positive.

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

**Example**

| SAS Statement | Result  *
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>----+----1</td>
</tr>
</tbody>
</table>
SAS Statement | Result
--- | ---
date = '17mar2012'd;
juldate = put(date,pdjulg4.);
put juldate $hex8.;

2012077F

See Also

Formats:
- “JULDAYw. Format” on page 133
- “JULIANw. Format” on page 134
- “PDJULIw. Format” on page 154

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

Informats:
- “JULIANw. Informat” on page 324
- “PDJULGw. Informat” on page 334
- “PDJULIw. Informat” on page 336

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

**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 one-byte representation of a two-digit integer that represents the century.
- **yy** is the one-byte representation of a two-digit integer that represents the year. The PDJULIw. 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 one-and-a-half byte representation of the three-digit integer that corresponds to the Julian day of the year, 1–365 (or 1–366 for leap years).
- **F** is the half byte that contains all binary 1s, which assigns the value as positive.

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

Example

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>date = '17mar2012'd;</em></td>
<td></td>
</tr>
<tr>
<td>juldate = put(date,pdjuli4.);</td>
<td>0112077F</td>
</tr>
<tr>
<td>put juldate $hex8.;</td>
<td></td>
</tr>
<tr>
<td><em>date = '31dec2013'd;</em></td>
<td></td>
</tr>
<tr>
<td>juldate = put(date,pdjuli4.);</td>
<td>0113365F</td>
</tr>
<tr>
<td>put juldate $hex8.;</td>
<td></td>
</tr>
</tbody>
</table>

See Also

**Formats:**
- “JULDAYw. Format” on page 133
- “JULIANw. Format” on page 134
- “PDJULGw. Format” on page 153

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

**Informats:**
- “JULIANw. Informat” on page 324
PERCENTw.d Format

Writes numeric values as percentages.

**Category:** 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 negative numbers, whether the number is negative or positive.

**d**

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

Range 0–31

Requirement must be less than w

**Details**

The 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**

    put @10 gain percent10.;
Value of x | Result
-------------|-------------
| 0.1 | 10%
| 1.2 | 120%
| -0.05 | (-5%)

See Also

Formats:
- “PERCENTNw.d Format” on page 157

PERCENTNw.d Format

Produces percentages, using a minus sign for negative values.

- **Category:** 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

PERCENTNw.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 PERCENTNw.d format multiplies negative values by 100, formats them the same as the BESTw.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 PERCENTNw.d format produces percents by using a minus sign instead of parentheses for negative values. The PERCENTw.d format produces percents by using parentheses for negative values.

Example

```sas
put x percentn10.;
```

<table>
<thead>
<tr>
<th>Value of x</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 156

PIBw.d Format

Writes positive integer binary (fixed-point) values.

Category: Numeric
Alignment: Left

See:

- “PIBw.d Format: UNIX” in SAS Companion for UNIX Environments
- “PIBw.d Format: Windows” in SAS Companion for Windows

Syntax

PIBw.d
**Syntax Description**

\( w \)

specifies the width of the output field.

Default 1

Range 1–8

\( d \)

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

Default 0

Range 0–31

**Details**

All values are treated as positive. PIB\(w.d\) writes positive integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

*Note:* Different operating environments store integer binary values in different ways. This concept is called byte ordering. For more information about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 8.

**Comparisons**

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIB\(w.d\) format treats all values as positive and includes the sign bit as part of the value.

- The PIB\(w.d\) format with a width of 1 results in a value that corresponds to the binary equivalent of the contents of a byte. A value that corresponds to the binary equivalent of the contents of a byte is useful if your data contains values between hexadecimal 80 and hexadecimal FF, where the high-order bit can be misinterpreted as a negative sign.

- The PIB\(w.d\) format is the same as the IB\(w.d\) format except that PIB\(w.d\) treats all values as positive values.

- The IB\(w.d\) and PIB\(w.d\) formats are used to write native format integers. (Native format enables you to read and write values that are created in the same operating environment.) The IBR\(w.d\) and PIBR\(w.d\) formats are used to write little endian integers in any operating environment.

To view a table that shows the type of format to use with big endian and little endian integers, see “Writing Data Generated on Big Endian or Little Endian Platforms” on page 8.

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

**Example**

\[ y = \text{put}(x, \text{pib1.}); \]

\[ \text{put } y \ \$\text{hex2.}; \]
The result is a hexadecimal representation of a one-byte binary number written in positive integer binary format, which occupies one column of the output field.

**See Also**

Formats:
- “PIBr.w.d Format” on page 160

**PIBr.w.d Format**

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

**Category:** Numeric

**Syntax**

PIBr.w.d

**Syntax Description**

\( w \)

specifies the width of the input field.

Default 1

Range 1–8

\( d \)

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

Default 0

Range 0–10

**Details**

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

**Note:** Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 8.
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 contain values between hexadecimal 80 and hexadecimal FF, where the high-order bit can be misinterpreted as a negative sign.

- On Intel and DEC operating environments, the 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 or Little Endian Platforms” on page 8.

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

Example

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

<table>
<thead>
<tr>
<th>Value of x</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 339

PKw.d Format

Writes data in unsigned packed decimal format.

Category: Numeric
Alignment: Left
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

\[PK_{w.d}\]

**Syntax Description**

- **w**: specifies the width of the output field.
  - Default: 1
  - Range: \(1–16\)

- **d**: specifies to multiply the number by \(10^d\). This argument is optional.
  - Default: 0
  - Range: \(0–10\)
  - Requirement: must be less than \(w\)

**Details**

Each byte of unsigned packed decimal data contains two digits.

**Comparisons**

The PK\(_{w.d}\) format is similar to the PD\(_{w.d}\) format except that PK\(_{w.d}\) does not write the sign in the low-order byte.

**Example**

\[y=put(x,pk4.);
put y $hex8.;\]

<table>
<thead>
<tr>
<th>Value of (x)</th>
<th>Result $\hex8.$</th>
</tr>
</thead>
<tbody>
<tr>
<td>(128)</td>
<td>00000128</td>
</tr>
</tbody>
</table>

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

Writes p-values.

Category: 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

PVALUEw.d

Syntax Description

w
specifies the width of the output field.
Default 6
Range 3–32

d
specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.
Default the minimum of 4 and w–2
Range 1–30
Restriction must be less than w

Comparisons

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

• if the value x is such that 0 <= x < 10^–d, x is printed as “<.0...01” with d-1 zeros
• missing values are printed as “.” unless you specify a different character by using the MISSING= system option

Example

put x pvalue6.4;

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0000000</td>
<td>-1.000</td>
</tr>
</tbody>
</table>
### QTRw. Format

Writes date values as the quarter of the year.

- **Category:** Date and Time
- **Alignment:** Right

#### Syntax

```
QTRw.
```

#### Syntax Description

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

  - **Default:** 1
  - **Range:** 1–32

#### Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>put date qtr.;</code></td>
<td>1</td>
</tr>
</tbody>
</table>

#### See Also

- “QTRRw. Format” on page 165
QTRRw. Format

Writers date values as the quarter of the year in Roman numerals.

- **Category:** Date and Time
- **Alignment:** Right

**Syntax**

QTRRw.

**Syntax Description**

w

specifies the width of the output field.

- **Default:** 3
- **Range:** 3–32

**Example**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date qtrr.;</td>
<td>III</td>
</tr>
</tbody>
</table>

**See Also**

- Formats:
  - “QTRw. Format” on page 164

RBw.d Format

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

- **Category:** Numeric
- **Alignment:** Left

**See:**

- “RBw.d Format: UNIX” in SAS Companion for UNIX Environments
- “RBw.d Format: Windows” in SAS Companion for Windows
- “RBw.d Format: z/OS” in SAS Companion for z/OS
Syntax

\textbf{RB}_{w.d}

\textbf{Syntax Description}

\textbf{w}

specifies the width of the output field.

Default 4

Range 2–8

\textbf{d}

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

Default 0

Range 0–10

\textbf{Details}

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

\textit{Note:} Different operating environments store real binary values in different ways. However, \texttt{RB}_{w.d} writes real binary values with consistent results in the same type of operating environment that you use to run SAS.

\textbf{CAUTION:}

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

\textbf{Comparisons}

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

\begin{table}
\begin{tabular}{|l|c|c|}
\hline
\textbf{Language} & \textbf{4 Bytes} & \textbf{8 Bytes} \\
\hline
SAS & \texttt{RB4.} & \texttt{RB8.} \\
\hline
Fortran & \texttt{REAL*4} & \texttt{REAL*8} \\
\hline
C & \texttt{float} & \texttt{double} \\
\hline
COBOL & \texttt{COMP-1} & \texttt{COMP-2} \\
\hline
IBM 370 assembler & \texttt{E} & \texttt{D} \\
\hline
\end{tabular}
\end{table}
Example

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

<table>
<thead>
<tr>
<th>Value of ( x )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>( --------+----)</td>
<td>( 1----+----2 )</td>
</tr>
<tr>
<td>128</td>
<td>4280000000000000</td>
</tr>
</tbody>
</table>

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

**ROMANw. Format**

Writes numeric values as roman numerals.

- **Category:** 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**

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

<table>
<thead>
<tr>
<th>Value of <code>year</code></th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>MMXII</td>
</tr>
</tbody>
</table>
S370FFw.d Format

Writes native standard numeric data in IBM mainframe format.

**Category:** Numeric

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

<table>
<thead>
<tr>
<th>Value of x</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 46
- “w.d Format” on page 192

S370FIBw.d Format

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

Category: Numeric
Alignment: Left

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 S370FIBw.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. S370FIBw.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 S370FIBw.d to write integer binary data in IBM mainframe format from data that are created in other operating environments.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 8.

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 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 or Little Endian Platforms” on page 8.

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

Example

```plaintext
y=put(x,m370fib4.);
put y $hex8.;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<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 170
- “S370FPIBw.d Format” on page 174

S370FIBUw.d Format

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

**Category:** Numeric  
**Alignment:** Left

**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 are created in other operating environments.

*Note:* Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 8.

**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, while 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 or Little Endian Platforms” on page 8.

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

**Example**

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

<table>
<thead>
<tr>
<th>Value of (x)</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 one-byte integer binary number. Each byte occupies one column of the output field.
See Also

Formats

• “S370FIBw.d Format” on page 169
• “S370FPBw.d Format” on page 174

S370FPDw.d Format

Writes packed decimal data in IBM mainframe format.

Category: Numeric
Alignment: Left
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

S370FPDw.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 S370FPDw.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:

<table>
<thead>
<tr>
<th>Language</th>
<th>Packed Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FPD4.</td>
</tr>
<tr>
<td>Language</td>
<td>Packed Decimal Notation</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------</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**

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

<table>
<thead>
<tr>
<th>Value of x</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  
**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.
  
  **Default:** 1  
  **Range:** 1–16

`d`

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

Use S370FPDUw.d in other operating environments to write unsigned packed decimal data in the same format as on an IBM mainframe computer.

Comparisons

- The S370FPDUw.d format is similar to the S370FPDw.d format except that the S370FPDw.d format always uses the absolute value instead of the signed value.
- The S370FPDUw.d format is equivalent to the COBOL notation PIC 9(n) PACKED-DECIMAL, where the n value is the number of digits.

Example

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

<table>
<thead>
<tr>
<th>Value of x</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 one byte of binary data, and each byte corresponds to one column of the output field.

---

S370FPIBw.d Format

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

**Category:** Numeric  
**Alignment:** Left

**Syntax**

S370FPIBw.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>4</td>
<td>1–8</td>
</tr>
</tbody>
</table>
$d$
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 are created in other operating environments.

Note: Different operating environments store integer binary values in different ways.
This concept is called byte ordering. For a detailed discussion about byte ordering,
see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian
Platforms” on page 8.

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 or Little Endian Platforms” on
page 8.

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

Example

```sas
y=put(x,m370fpib1.);
put y $hex2.;;
```

<table>
<thead>
<tr>
<th>Value of x</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 one-byte binary number written in positive integer binary format, which occupies one column of the output field.

See Also

Formats:
**S370FRBw.d Format**

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

**Category:** Numeric

**Alignment:** Left

### Syntax

\[ \text{S370FRB} \text{w}.d \]

### Syntax Description

\( w \)

specifies the width of the output field.

**Default:** 4

**Range:** 2–8

\( d \)

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

**Default:** 0

**Range:** 0–10

### Details

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

Use \text{S370FRBw.d} in other operating environments to write floating-point binary data in the same format as on an IBM mainframe computer.

### Comparisons

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

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
<tr>
<td>Fortran</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-1</td>
<td>COMP-2</td>
</tr>
</tbody>
</table>
The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one column of the output field.

**S370FZDw.d Format**

Writes zoned decimal data in IBM mainframe format.

**Category:** Numeric  
**Alignment:** Left  
**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**

S370FZDw.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

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

<table>
<thead>
<tr>
<th>Value of (x)</th>
<th>Result (')</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>F1F2C3</td>
</tr>
<tr>
<td>-123</td>
<td>F1F2D3</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 one byte of binary data, and each byte corresponds to one column of the output field.

---

S370FZDL\(w.d\) Format

Writes zoned decimal leading–sign data in IBM mainframe format.

- **Category:** Numeric
- **Alignment:** Left
- **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

\textit{w}

specifies the width of the output field.

\textbf{Default} \hspace{1em} 8

\textbf{Range} \hspace{1em} 1–32

\textit{d}

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

\textbf{Default} \hspace{1em} 0

\textbf{Range} \hspace{1em} 0–31

Details

Use \texttt{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 \texttt{S370FZDLw.d} format is similar to the \texttt{S370FZDw.d} format except that the \texttt{S370FZDLw.d} format displays the sign of the number in the first byte of the formatted output.

- The \texttt{S370FZDLw.d} format is equivalent to the COBOL notation \texttt{PIC S9(n) DISPLAY SIGN LEADING}, where the \texttt{n} value is the number of digits.

Example

\begin{verbatim}
y=put(x,s370fzdl3.);
put y $hex6.;
\end{verbatim}

\begin{center}
\begin{tabular}{|c|c|}
\hline
\textbf{Value of \texttt{x}} & \textbf{Result \texttt{*}} \\
\hline
123 & C1F2F3 \\
\hline
-123 & D1F2F3 \\
\hline
\end{tabular}
\end{center}

\texttt{*} \hspace{1em} The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one column of the output field.

\section*{S370FZDSw.d Format}

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

\textbf{Category:} Numeric
Syntax

S370FZDS_{w,d}

Syntax Description

\( w \)

specifies the width of the output field.

Default 8
Range 2–32

\( d \)

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

Default 0
Range 0–31

Details

Use S370FZDS_{w,d} in other operating environments to write zoned decimal separate leading-sign data in the same format as on an IBM mainframe computer.

Comparisons

- The S370FZDS_{w,d} format is similar to the S370FZDL_{w,d} format except that the S370FZDS_{w,d} format does not embed the sign of the number in the zoned output.
- The S370FZDS_{w,d} format is equivalent to the COBOL notation PIC S9(n) DISPLAY SIGN LEADING SEPARATE, where the \( n \) value is the number of digits.

Example

\[
y=put (x,s370fzs4.);
puy y $hex8.;
\]

| Value of \( x \) | Result *
<table>
<thead>
<tr>
<th></th>
<th></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 one 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.
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 the n value is the number of digits.

Example

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

<table>
<thead>
<tr>
<th>Value of x</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 one 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  
**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 the $n$ value is the number of digits.

### Example

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

Writes Social Security numbers.

Category: Numeric

Syntax

SSNw.

Syntax Description

w

specifies the width of the output field.

Default 11

Restriction w must be 11

Details

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

Example

    put id ssn.;

<table>
<thead>
<tr>
<th>Value of id</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>----- ------</td>
<td>----------------</td>
</tr>
<tr>
<td>263878439</td>
<td>263-87-8439</td>
</tr>
</tbody>
</table>
**TIMEw.d Format**

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

**Category:** 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

**TIMEw.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 TIMEw.d format writes SAS time values in the form `hh:mm:ss.ss`:

- **hh** is an integer.
  - *Note:* If `hh` is a single digit, TIMEw.d places a leading blank before the digit. For example, the TIMEw.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 TIMEw.d format is similar to the HHMMw.d format except that TIMEw.d includes seconds.

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

Examples

Example 1
This example uses the input value of 59083, which is the SAS time value that corresponds to 4:24:43 p.m.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put begin time.;</td>
<td>16:24:43</td>
</tr>
</tbody>
</table>

Example 2
This example uses the input value of 32083, which is the SAS time value that corresponds to 8:54:43 a.m.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put begin time.;</td>
<td>08:54:43</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “HHMMw.d Format” on page 126
- “HOURw.d Format” on page 128
- “MMSSw.d Format” on page 141
- “TODw.d Format” on page 187

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

Informats:
- “TIMEw. Informat” on page 369
TIMEAMPM\(w.d\) Format

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

**Category:** Date and Time  
**Alignment:** Right  
**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**

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

**Syntax Description**

\(w\)

specifies the width of the output field.

Default: 11  
Range: 2–20

\(d\)

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

Default: 0  
Range: 0–19

Requirement: must be less than \(w\)

**Details**

The TIMEAMPM\(w.d\) format writes SAS time values and SAS datetime values in the form \(hh:mm:ss.ss\) with AM or PM, where

\(hh\)

is an integer that represents the hour.

\(mm\)

is an integer that represents the minutes.

\(ss.ss\)

is the number of seconds to two decimal places.

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

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

- The TIMEAMPMMw.d format is similar to the TIMEMw.d format except, that TIMEAMPMMw.d prints AM or PM at the end of the time.
- TIMEw.d writes hours greater than 23:59:59 PM, and TIMEAMPMw.d does not.

Example

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

<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 timeampm7.;</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:
- “TIMEw.d Format” on page 184

TODw.d Format

Writers SAS time values and the time portion of SAS datetime values in the form hh:mm:ss.ss.

- **Category:** 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

TODw.d

**Syntax Description**

w

specifies the width of the output field.
Default 8
Range 2–20
Tip SAS writes a zero for a zero hour if the specified width is sufficient. For example, the times 02:30 or 00:30 have a zero in the hour digits.

\[ d \]

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

Default 0
Range 0–19
Requirement must be less than \( w \)

Details

The TOD\(_w.d\) format writes SAS time and datetime values in the form hh:mm:ss.ss:

\[ hh \]

is an integer that represents the hour.

\[ mm \]

is an integer that represents the minutes.

\[ ss.ss \]

is the number of seconds to two decimal places.

Comparisons

The TOD\(_w.d\) format writes a leading zero for a single-hour digit. The TIME\(_w.d\) format and the HHMM\(_w.d\) format write a leading blank for a single-hour digit.

Examples

**Example 1**
In this example, the SAS datetime value 1661437223 corresponds to August 24, 2012 at 2:20:23 p.m.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin = '1:30't; put begin tod5.;</td>
<td>01:30</td>
</tr>
<tr>
<td>begin = 1661437223; put begin tod9.;</td>
<td>14:20:23</td>
</tr>
</tbody>
</table>

**Example 2**
In this example, the SAS time value 32083 corresponds to 8:54:43 a.m.
begin = 32083;
put begin tod9.;

See Also

Formats:
- “HHMMw.d Format” on page 126
- “TIMEw.d Format” on page 184
- “TIMEAMPMw.d Format” on page 186

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

Informats:
- “TIMEw. Informat” on page 369

VAXRBw.d Format

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

Category: 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 VAXRB\textit{w.d} format to write data in native VAX or VMS floating-point notation.

Comparisons
If you use SAS that is running under VAX or VMS, then the VAXRB\textit{w.d} and the RB\textit{w.d} formats are identical.

Example
\begin{verbatim}
x=1;
y=put(x,vaxrb8.);
put y=$hex16.;
\end{verbatim}

\begin{tabular}{|c|c|}
\hline
Value of x & Result \\
\hline
1 & 8040000000000000 \\
\hline
\end{tabular}

* The result is the hexadecimal representation for the integer.

---

### VMSZN\textit{w.d} Format
Generates VMS and MicroFocus COBOL zoned numeric data.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
</tbody>
</table>

#### Syntax
\texttt{VMSZN\textit{w.d}}

#### Syntax Description
\texttt{w}

\textit{w} specifies the width of the output field.

Default 1

Range 1–32

\texttt{d}

\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\textit{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. The following table shows the special characters that are used by the VMSZNw.d format.

<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>

If the value to be formatted is too large to fit in a field of the specified width, then the VMSZNw.d format does the following:

- For positive values, it sets the output to the largest positive number that fits in the given width.
- For negative values, it sets the output to the negative number of greatest magnitude that fits in the given width.

**Example**

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>x=1234; put x vmszn4.;</td>
<td>1234</td>
</tr>
<tr>
<td>x=1234; put x vmszn5.1;</td>
<td>12340</td>
</tr>
<tr>
<td>x=1234; put x vmszn6.2;</td>
<td>123400</td>
</tr>
<tr>
<td>x=-1234; put x vmszn5.;</td>
<td>0123t</td>
</tr>
</tbody>
</table>
See Also

Formats:
  • “ZDw.d Format” on page 222

Informats:
  • “VMSZNw.d Informat” on page 374

\( w.d \) Format

Writes standard numeric data one digit per byte.

- **Category:** 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 BEST\( w \) format. The \( w.d \) format writes negative numbers with leading minus signs. In addition, \( w.d \) right aligns before writing and pads the output with leading blanks.
Comparisons

The \( Zw.d \) format is similar to the \( w.d \) format except that \( Zw.d \) pads right-aligned output with 0s instead of blanks.

Example

\[
\text{put @7 x 6.3;}
\]

<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 \( \text{day-of-week, month-name dd, yy (or yyyy)} \).

**Category:** Date and Time  
**Alignment:** Right

**Syntax**

\[
\text{WEEKDATEw}.
\]

**Syntax Description**

\( w \)

specifies the width of the output field.

- **Default:** 29  
- **Range:** 3–37

**Details**

The \( \text{WEEKDATEw} \) format writes SAS date values in the form \( \text{day-of-week, month-name dd, yy (or yyyy)} \):

- \( dd \)  
  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 \textit{dd} before the month's name.

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date weekdate3.;</td>
<td>Thu</td>
</tr>
<tr>
<td>put date weekdate9.;</td>
<td>Thursday</td>
</tr>
<tr>
<td>put date weekdate15.;</td>
<td>Thu, Jun 14, 12</td>
</tr>
<tr>
<td>put date weekdate17.;</td>
<td>Thu, Jun 14, 2012</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DATEw. Format” on page 89
- “DDMMYYw. Format” on page 95
- “MMDDYYw. Format” on page 137
- “TODw.d Format” on page 187
- “WEEKDATXw. Format” on page 195
- “YYMMDDw. Format” on page 209

Functions:

- “JULDATE Function” in \textit{SAS Functions and CALL Routines: Reference}
- “MDY Function” in \textit{SAS Functions and CALL Routines: Reference}
- “WEEKDAY Function” in \textit{SAS Functions and CALL Routines: Reference}

Informats:

- “DATEw. Informat” on page 300
- “DDMMYYw. Informat” on page 303
- “MMDDYYw. Informat” on page 327
- “YYMMDDw. Informat” on page 385
**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*).

**Category:** Date and Time  
**Alignment:** Right

### Syntax

WEEKDATXw.

### Syntax Description

**w**

specifies the width of the output field.

- **Default:** 29
- **Range:** 3–37

### Details

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

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

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

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

### Comparisons

The WEEKDATEw. format is the same as the WEEKDATXw. format, except that WEEKDATEw. prints *dd* after the month's name.

The WEEKDATXw. format is the same as the DTWKDATXw. format, except that DTWKDATXw. expects a datetime value as input.

### Example

The example table uses the input value of 19046, which is the SAS date value that corresponds to February 23, 2012.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date weekdatx.;</td>
<td>Thursday, 23 February 2012</td>
</tr>
</tbody>
</table>
See Also

Formats:
- “DATEw. Format” on page 89
- “DDMMYYw. Format” on page 95
- “DTWKDATXw. Format” on page 104
- “MMDDYYw. Format” on page 137
- “TODw.d Format” on page 187
- “WEEKDATEw. Format” on page 193
- “YYMMDDw. Format” on page 209

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 300
- “DDMMYYw. Informat” on page 303
- “MMDDYYw. Informat” on page 327
- “YYMMDDw. Informat” on page 385

---

**WEEKDAYw. Format**

Writes date values as the day of the week.

**Category:** Date and Time  
**Alignment:** Right

**Syntax**

```
WEEKDAY
```

**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

The example table uses the input value of 19025, which is the SAS date value that corresponds to February 2, 2012.

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

See Also

Formats:

- “DOWNAMEw. Format” on page 101

WEEKUw. Format

Writes a week number in decimal format by using the U algorithm.

Category: Date and Time

Alignment: Left

Syntax

WEEKUw.

Syntax Description

w

specifies the width of the output field.

Default 11

Range 3–200

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 zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

Refer to the following table for widths, formats, and examples:
Comparisons

The WEEKW format writes the week number of the year as a decimal number in the range 00–53, with Monday as the first day of week 1. The WEEKU 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 zero.

Example

sasdate = '31JAN2012'd;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>v=put(sasdate,weeku3.);</td>
<td>W05</td>
</tr>
<tr>
<td>w=put(sasdate,weeku5.);</td>
<td>12W05</td>
</tr>
<tr>
<td>x=put(sasdate,weeku7.);</td>
<td>12W0503</td>
</tr>
<tr>
<td>y=put(sasdate,weeku9.);</td>
<td>2012W0503</td>
</tr>
<tr>
<td>z=put(sasdate,weeku11.);</td>
<td>2012-W05-03</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WEEKVw. Format” on page 199
- “WEEKWw. Format” on page 200

Functions:
- “WEEK Function” in SAS Functions and CALL Routines: Reference
WEEKVw. Format

WEEKVw. Format writes a week number in decimal format by using the V algorithm.

**Category:** Date and Time

**Alignment:** Left

**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>3–200</td>
</tr>
</tbody>
</table>

**Details**

The WEEKVw. format writes the various formats depending on the specified width. Algorithm V calculates the SAS date value, with the number-of-the-week value represented as a decimal number in the range 01–53, with a leading zero and maximum value of 53. Weeks begin on a Monday and week 1 of the year is the week that includes both January 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.

Refer to the following table for 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>12W01</td>
</tr>
<tr>
<td>7-8</td>
<td>yyWwwd</td>
<td>12W0101</td>
</tr>
<tr>
<td>9-10</td>
<td>yyyyWwwdd</td>
<td>2012W0101</td>
</tr>
<tr>
<td>11-200</td>
<td>yyyy-Www-dd</td>
<td>2012-W01-01</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 zero.

Example

sasdate='31JAN2012'd;

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>v=put(sasdate,weekv3.);</td>
<td>W05</td>
</tr>
<tr>
<td>w=put(sasdate,weekv5.);</td>
<td>12W01</td>
</tr>
<tr>
<td>x=put(sasdate,weekv7.);</td>
<td>12W0502</td>
</tr>
<tr>
<td>y=put(sasdate,weekv9.);</td>
<td>2012W0502</td>
</tr>
<tr>
<td>z=put(sasdate,weekv11.);</td>
<td>2012-W05-02</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WEEKUw. Format” on page 197
- “WEEKWw. Format” on page 200

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

Informats:
- “WEEKUw. Informat” on page 377
- “WEEKVw. Informat” on page 379
- “WEEKWw. Informat” on page 381

WEEKWw. Format

Writes a week number in decimal format by using the W algorithm.

Category: Date and Time
Syntax

WEEKWw.

Syntax Description

w

specifies the width of the output field.

Default 11
Range 3–200

Details

The WEEKWw. 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 zero 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>12W01</td>
</tr>
<tr>
<td>7–8</td>
<td>yyWwdd</td>
<td>12W0101</td>
</tr>
<tr>
<td>9–10</td>
<td>yyyyWwdd</td>
<td>2012W0101</td>
</tr>
<tr>
<td>11–200</td>
<td>yyyy-Www-dd</td>
<td>2012-W01-01</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 zero.

Example

sasdate = '31JAN2012'd;
### See Also

**Formats:**
- “WEEKUw. Format” on page 197
- “WEEKVw. Format” on page 199

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

**Informats:**
- “WEEKUw. Informat” on page 377
- “WEEKVw. Informat” on page 379
- “WEEKWw. Informat” on page 381

---

**WORDDATEw. Format**

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

**Category:** Date and Time

**Alignment:** Right

---

**Syntax**

WORDDATEw.

**Syntax Description**

*w*

specifies the width of the output field.

**Default** 18
Range 3–32

Details

The WORDDATEw. format writes SAS date values in the form month-name dd, yyyy:

- dd is an integer that represents the day of the month.
- yyyy is a four-digit integer that represents the year.

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

Comparisons

The WORDDATEw. format is the same as the WORDDATXw. format except that WORDDATXw. prints dd before the month's name.

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put term worddate3.;</td>
<td>Jun</td>
</tr>
<tr>
<td>put term worddate9.;</td>
<td>--+-+---1-+---+----2</td>
</tr>
<tr>
<td>put term worddate12.;</td>
<td>June</td>
</tr>
<tr>
<td>put term worddate20.;</td>
<td>Jun 14, 2012</td>
</tr>
<tr>
<td>put term worddate20.;</td>
<td>--+-+---1-+---+----2</td>
</tr>
<tr>
<td>put term worddate20.;</td>
<td>June 14, 2012</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WORDDATXw. Format” on page 203

WORDDATXw. Format

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

- Category: Date and Time
- Alignment: Right

Syntax

WORDDATXw.
Syntax Description

w
specifies the width of the output field.

Default  18
Range      3–32

Details

The WORDDATXw. format writes SAS date values in the form \textit{dd month-name, yyyy}:

\textit{dd}

is an integer that represents the day of the month. For days 1–9, the leading 0 is not displayed.

\textit{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 \textit{dd} after the month's name.

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put term worddatx.;</td>
<td>5 March 2012</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WORDDATEw. Format” on page 202

WORDFw. Format

Writes numeric values as words with fractions that are shown numerically.

\begin{tabular}{|c|c|}
\hline
Category & Numeric \\
\hline
Alignment & Left \\
\hline
\end{tabular}
Syntax

\texttt{WORDF}_{w}.

\textit{Syntax Description}

\texttt{w}

specifies the width of the output field.

Default 10

Range 5–32767

Details

The \texttt{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 into the specified field, it is truncated on the right and the last character is printed as an asterisk.

Comparisons

The \texttt{WORDF}_{w}. format is similar to the \texttt{WORDS}_{w}. format except that \texttt{WORDF}_{w}. prints fractions as numbers instead of words.

Example

\begin{verbatim}
    put price wordf15.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Value of \texttt{price}</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\hline</td>
</tr>
<tr>
<td>2.5</td>
<td>two and 50/100</td>
</tr>
</tbody>
</table>

See Also

Formats:

- "\textit{WORDS}_{w}. Format" on page 205
Syntax

WORDS\textsubscript{w}.

Syntax Description

\texttt{w}

specifies the width of the output field.

Default

10

Range

5–32767

Details

You can use the WORD\textsubscript{w} format to print checks with the amount written out below the payee line.

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

Comparisons

The WORD\textsubscript{w} format is similar to the WORDF\textsubscript{w} format except that WORD\textsubscript{w} prints fractions as words instead of numbers.

Example

put price words23.;

<table>
<thead>
<tr>
<th>Value of price</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>two and ten hundredths</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “WORDF\textsubscript{w} Format” on page 204

YEAR\textsubscript{w} Format

Writes date values as the year.

Category: Date and Time

Alignment: Right
Syntax

YEARw.

Syntax Description

w

specifies the width of the output field.

Default 4

Range 2–32

Tip If w is less than 4, the last two digits of the year are printed. Otherwise, the year value is printed as four digits.

Details

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

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date year2.;</td>
<td>12</td>
</tr>
<tr>
<td>put date year4.;</td>
<td>2012</td>
</tr>
</tbody>
</table>

See Also

Formats:

• “DTYEARw. Format” on page 106

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 2 or 4 digits.

Category: Date and Time

Alignment: Right
Syntax

YYMMw.

Syntax Description

w

specifies the width of the output field.

Default 7

Range 5–32

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

Details

The 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

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date yymm5.;</td>
<td>12M10</td>
</tr>
<tr>
<td>put date yymm6.;</td>
<td>12M10</td>
</tr>
<tr>
<td>put date yymm.;</td>
<td>2012M10</td>
</tr>
<tr>
<td>put date yymm7.;</td>
<td>2012M10</td>
</tr>
<tr>
<td>put date yymm10.;</td>
<td>2012M10</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “MMYYw. Format” on page 142
YYMMDw. Format

Writes date values in the form yymmd or <yy>yy-mm-dd, where a hyphen is the separator and the year appears as either 2 or 4 digits.

**Category:** Date and Time  
**Alignment:** Right

**Syntax**

\texttt{YYMMDw.}

**Syntax Description**

\texttt{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>2–10</td>
</tr>
</tbody>
</table>

**Interaction**

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

**Details**

The YYMMDw. format writes SAS date values in one of the following forms:

\texttt{yymmd}

\texttt{<yy>yy-mm-dd}

where

\texttt{<yy>yy}

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

\_\_\_\_

is the separator.

\texttt{mm}

is an integer that represents the month.

\texttt{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 YYMMDDx. format.

**Example**

The following examples use the input value of 19086, which is the SAS date value that corresponds to April 3, 2012.
YYMMDXw. Format

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

YYYYMMDD\textsubscript{xw}.

**Syntax Description**

\(x\)

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

- **B**: separates with a blank.
- **C**: separates with a colon.
- **D**: separates with a hyphen.
- **N**: indicates no separator.
- **P**: separates with a period.
- **S**: separates with a slash.

\(w\)

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>2–10</td>
</tr>
<tr>
<td>Interactions</td>
<td>When (w) has a value of from 2 to 5, the date appears with as much of the year and the month. When (w) is 7, the date appears as a two-digit year without separators. When (x) has a value of N, the width range is 2–8.</td>
</tr>
</tbody>
</table>

**Details**

The YYYYMMDD\textsubscript{xw} format writes SAS date values in one of the following forms:

- **yyymmdd**
- \(<\\texttt{yy}\\texttt{mm}\\texttt{xx}\\texttt{dd}\)**

where

- \(<\\texttt{yy}\\texttt{mm}\\texttt{xx}\\texttt{dd}\) is a two-digit or four-digit integer that represents the year.

- \(x\) is a specified separator.
is an integer that represents the month.

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

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put day yymmddc5.;</td>
<td>12:05</td>
</tr>
<tr>
<td>put day yymmdd8.;</td>
<td>12-05-14</td>
</tr>
<tr>
<td>put day yymmddp10.;</td>
<td>2012.05.14</td>
</tr>
<tr>
<td>put day yymmddn8.;</td>
<td>20120514</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 89
- “DDMMYYxw. Format” on page 96
- “MMDDYYxw. Format” on page 139
- “YYMMDDw. Format” on page 209

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 385

YYMMxw. Format

Writes date values in the form <yy>yyymm or <yy>yy-mm. The x in the format name represents the special character that separates the year and the month. This special character can be a hyphen (-), period (.), slash (/), colon (:), or no separator. The year can be either two or four digits.
Syntax
YYMMxw.

Syntax Description

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

\( C \)
separates with a colon.

\( D \)
separates with a hyphen.

\( N \)
indicates no separator.

\( P \)
separates with a period.

\( S \)
separates with a forward slash.

\( w \)
specifies the width of the output field.

Default 7

Range 5–32

Interactions

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

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

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

Details

The YYMMxw. format writes SAS date values in one of the following forms:

\(<yy>yyymm\)

\(<yy>yyXmm\)

where

\(<yy>yy\)

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

\( x \)

is a specified separator.

\( mm \)

is an integer that represents the month.
Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date yymmc5.;</td>
<td>12:05</td>
</tr>
<tr>
<td>put date yymmd.;</td>
<td>2012-05</td>
</tr>
<tr>
<td>put date yymmn4.;</td>
<td>1205</td>
</tr>
<tr>
<td>put date yymmp8.;</td>
<td>2012.05</td>
</tr>
<tr>
<td>put date yymms10.;</td>
<td>2012/05</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “MMYYxw. Format” on page 143
- “YYMMw. Format” on page 207

YYMONw. Format

Writes date values in the form yymm or yyyyymm.

Category: Date and Time
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>yymm:
<yy>yy
is a two-digit or four-digit integer that represents the year.

mmm
is the name of the month, abbreviated to three characters.

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date yymon6.;</td>
<td>02JUN</td>
</tr>
<tr>
<td>put date yymon7.;</td>
<td>2012JUN</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “MMYYw. Format” on page 142

**YYQw. Format**

Writes date values in the form <yy>yyQq, where Q is the separator, the year appears as either 2 or 4 digits, and q is the quarter of the year.

**Category:** Date and Time

**Alignment:** Right

**Syntax**

**YYQw.**

**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>6</td>
<td>4–32</td>
<td>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.</td>
</tr>
</tbody>
</table>
Details
The YYQw. format writes SAS date values in the form <yy>yyQq:

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

Q
 is the character separator.

q
 is an integer (1,2,3, or 4) that represents the quarter of the year.

Example
The following examples use the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date yyq4.;</td>
<td>12Q2</td>
</tr>
<tr>
<td>put date yyq5.;</td>
<td>12Q2</td>
</tr>
<tr>
<td>put date yyq.;</td>
<td>2012Q2</td>
</tr>
<tr>
<td>put date yyq6.;</td>
<td>2012Q2</td>
</tr>
<tr>
<td>put date yyq10.;</td>
<td>2012Q2</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “YYQxw. Format” on page 216
- “YYQRw. Format” on page 218

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, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.

Category: Date and Time
Alignment: Right

Syntax
YYQxw.
**Syntax Description**

\( x \)

identifies a separator or specifies that no separator appear between the year and the quarter. Valid values for \( x \) are:

- **C** separates with a colon
- **D** separates with a hyphen
- **N** indicates no separator
- **P** separates with a period
- **S** separates with a forward slash.

\( w \)

specifies the width of the output field.

**Details**

The YYQ\( xw \). format writes SAS date values in one of the following forms:

\(<yy>\text{yy}q\)

\(<yy>\text{yy}xq\)

where

- \( <yy>\text{yy} \) is a two-digit or four-digit integer that represents the year.
- \( x \) is a specified separator.
- \( q \) is an integer (1,2,3, or 4) that represents the quarter of the year.

**Example**

The following examples use the input value of 19188, which is the SAS date value that corresponds to July 14, 2012.
SAS Statement | Result
---|---
put date yyqc4.; | 12:3
put date yyqd.; | 2012-3
put date yyqn3.; | 123
put date yyqp6.; | 2012.3
put date yyqs8.; | 2012/3

See Also

Formats:
- “YYQw. Format” on page 215
- “YYQRxw. Format” on page 219

YYQRw. Format

Writes date values in the form <yy>yyQqr, where Q is the separator, the year appears as either 2 or 4 digits, and qr is the quarter of the year expressed in roman numerals.

Category: 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>yy

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

$qr$ is a roman numeral (I, II, III, or IV) that represents the quarter of the year.

**Example**

The following examples use the input value of 19158, which is the SAS date value that corresponds to June 14, 2012.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date yyqr6.;</td>
<td>12QII</td>
</tr>
<tr>
<td>put date yyqr7.;</td>
<td>2012QII</td>
</tr>
<tr>
<td>put date yyqr.;</td>
<td>2012QII</td>
</tr>
<tr>
<td>put date yyqr8.;</td>
<td>2012QII</td>
</tr>
<tr>
<td>put date yyqr10.;</td>
<td>2012QII</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “YYQw. Format” on page 215
- “YYQRxw. Format” on page 219

---

**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 that separates the year and the quarter or the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits and qr is the quarter of the year expressed in roman numerals.

**Category:** Date and Time

**Alignment:** Right

**Syntax**

YYQRxw.
**Syntax Description**

$x$

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

- **C** separates with a colon.
- **D** separates with a hyphen.
- **N** indicates no separator.
- **P** separates with a period.
- **S** separates with a forward slash.

$w$

specifies the width of the output field.

**Default**

8

**Range**

6–32

**Interactions**

When $x$ is set to $N$, no separator is specified. The width range is then 5–32, and the default changes to 7.

When the value of $w$ is too small to write a four-digit year, the date appears with only the last two digits of the year.

**Details**

The YYQR$xw$. format writes SAS date values in one of the following forms:

- `<yy>yyqr`
- `<yy>yyxxqr`

where

- `<yy>yy`
  is a two-digit or four-digit integer that represents the year.
- $x$
  is a specified separator.
- $qr$
  is a roman numeral (I, II, III, or IV) that represents the quarter of the year.

**Example**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SAS Statement | Result
---|---
put date yyqrc6.; | 12:II
put date yyqrd.; | 2012-II
put date yyqrn5.; | 12II
put date yyqrp8.; | 2012.II
put date yyqrs10.; | 2012/II

See Also

Formats:
- “YYQxw. Format” on page 216
- “YYQRw. Format” on page 218

Zw.d Format

Writes standard numeric data with leading 0s.

**Category:** 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.

**Default** 1  
**Range** 1–32  
**Tip** Allow enough space to write the value, the decimal point, and a minus sign, if necessary.

\[ d \]

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

The \( Z_{w.d} \) format writes standard numeric values one digit per byte and fills in 0s to the left of the data value.

The \( Z_{w.d} \) format rounds to the nearest number that fits into the output field. If \( w.d \) is too large to fit, SAS might shift the decimal to the BEST\( w \). format. The \( Z_{w.d} \) format writes negative numbers with leading minus signs. In addition, it right aligns before writing and pads the output with leading zeros.

Comparisons

The \( Z_{w.d} \) format is similar to the \( w.d \) format except that \( Z_{w.d} \) pads right-aligned output with 0s instead of blanks.

Example

```latex
put &5 seqnum z8.;
```

<table>
<thead>
<tr>
<th>Value of seqnum</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>00001350</td>
</tr>
</tbody>
</table>

ZD\( w.d \) Format

Writes numeric data in zoned decimal format.

**Category:** Numeric

**Alignment:** Left

**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:**
- "ZD\( w.d \) Format: UNIX" in SAS Companion for UNIX Environments
- "ZD\( w.d \) Format: Windows" in SAS Companion for Windows
- "ZD\( w.d \) Format: z/OS" in SAS Companion for z/OS

**Syntax**

\( ZD_{w.d} \)
Syntax Description

\( w \)

specifies the width of the output field.

Default 1

Range 1–32

\( d \)

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

Default 0

Range 0–31

Details

The zoned decimal format is similar to standard numeric format in that every digit requires one byte. However, the value's sign is in the last byte, along with the last digit.

Note: Different operating environments store zoned decimal values in different ways. However, the 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

```plaintext
y=put(x, zd4.);
put y $hex8.;
```

<table>
<thead>
<tr>
<th>Value of ( x )</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>120</td>
<td>F0F1F2C0</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 ................................................................. 227

Chapter 4
Dictionary of Informats ..................................................... 245
Chapter 3
About Informats

Definition of Informats

An informat is a type of SAS language element that applies a pattern to or executes instructions for a data value to be read as input. Types of informats correspond to the data's type: numeric, character, date, time, or timestamp. The ability to create user-defined informats is also supported. Examples of SAS informats are BINARY, DATE, and COMMA. For example, the following value contains a dollar sign and commas:

$1,000,000

To remove the dollar sign ($) and commas (,) before storing the numeric value 1000000 in a variable, read this value with the COMMA11 informat.
Unless you explicitly define a variable first, SAS uses the informat to determine whether the variable is numeric or character. SAS also uses the informat to determine the length of character variables.

Syntax

SAS informats have the following form:

```text
<$informat<w>.<d$>
```

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

`informat` names the informat. The informat is a SAS informat or a user-defined informat that was previously defined with the INVALUE statement in PROC FORMAT. “FORMAT” in Base SAS Procedures Guide.

`w` specifies the informat width, which for most informats is the number of columns in the input data.

`d` specifies an optional decimal scaling factor in the numeric informats. SAS divides the input data by 10 to the power of $d$.

Note: Even though SAS can read up to 32 digits when you specify some numeric informats, numbers with more than 15 significant digits might lose precision due to the limitations of the eight-byte floating-point representation used by most computers.

Informats always contain a period (.) as a part of the name. If you omit the $w$ and $d$ values from the informat, SAS uses default values. If the data contain decimal points, SAS ignores the $d$ value and reads the number of decimal places that are actually in the input data.

If the informat width is too narrow to read all the columns in the input data, you might get unexpected results. The problem frequently occurs with the date and time informats. You must adjust the width of the informat to include blanks or special characters between the day, month, year, or time. For more information about date and time values, see “Dates, Times, and Intervals” in SAS Language Reference: Concepts.

When a problem occurs with an informat, SAS writes a note to the SAS log and assigns a missing value to the variable. Problems occur if you use an incompatible informat, such as a numeric informat to read character data, or if you specify the width of a date and time informat that causes SAS to read a special character in the last column.

Using Informats

Ways to Specify Informats

Overview of Specifying Informats

You can specify informats in the following ways:
• in an INPUT statement
• with the INPUT, INPUTC, and INPUTN functions
• in an INFORMAT statement in a DATA step or a PROC step
• in an ATTRIB statement in a DATA step or a PROC step

**INPUT Statement**

The INPUT statement with an informat after a variable name is the simplest way to read values into a variable. For example, the following INPUT statement uses two informats:

```sas
input @15 style $3. @21 price 5.2;
```

The $w.d character informat reads values into the variable STYLE. The w.d numeric informat reads values into the variable PRICE.

For a complete discussion of the INPUT statement, see “INPUT Statement” in *SAS Statements: Reference*.

**INPUT Function**

The INPUT function converts a SAS character expression using a specified informat. The informat determines whether the resulting value is numeric or character. Thus, the INPUT function is useful for converting data. 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 DATEw 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 details about using the colon (:) modifier, see “INPUT Statement, List” in *SAS Statements: Reference*. Therefore, SAS uses a modified list input to read the variable so that

- the w value in an informat does not determine column positions or input field widths in an external file
- the blanks that are embedded in input data are treated as delimiters unless you change the DLM= or DLMSTR= option in an INFILE statement
- for character informats, the w value in an informat specifies the length of character variables
for numeric informats, the \( w \) value is ignored

for numeric informats, the \( d \) value in an informat behaves in the usual way for numeric informats.

If you have coded the INPUT statement to use another style of input, such as formatted input or column input, that style of input is not used when you use the INFORMAT statement.

See “INPUT Statement, List” in SAS Statements: Reference for more information about how to use modified list input to read data.

Note: Anytime a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option in either ASCII or EBCDIC environments. For example, when you read an EBCDIC text file on an ASCII platform, it is recommended that you specify the ENCODING= option in the FILENAME or INFILE statement. However, if you use the DSD and the DLM= or DLMSTR= options in the FILENAME or INFILE statement, the ENCODING= option is a requirement because these options require certain characters in the session encoding (such as quotation marks, commas, and blanks). The use of encoding-specific informats should be reserved for use with true binary files. That is, they contain both character and non-character fields.

ATTRIB Statement

The ATTRIB statement can also associate an informat, as well as other attributes, with one or more variables. For example, in the following statements, the ATTRIB statement associates the DATE\( w \) informat with the variables Birthdate and Interview:

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

An informat that is associated by using the INFORMAT= option in the ATTRIB statement behaves like an informat that you specify with a colon (:) format modifier in an INPUT statement. For details about using the colon (:) modifier, see “INPUT Statement, List” in SAS Statements: Reference. Therefore, SAS uses a modified list input to read the variable in the same way as it does for the INFORMAT statement.

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

Permanent versus Temporary Association

When you specify an informat in an INPUT statement, SAS uses the informat to read input data values during that DATA step. SAS, however, does not permanently associate the informat with the variable. To permanently associate an informat with a variable, use an INFORMAT statement or an ATTRIB statement. SAS permanently associates an informat with the variable by modifying the descriptor information in the SAS data set.

User-Defined Informats

In addition to the informats that are supplied with Base SAS software, you can create your own informats. In Base SAS software, PROC FORMAT enables you to create your own informats and formats for both character and numeric variables. For more information about user-defined informats, see “FORMAT” in Base SAS Procedures Guide.

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

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

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

<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: one byte, two bytes, or four 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 the SAS System, the following platforms are considered big endian: IBM mainframe, HP-UX, AIX, Solaris on SPARC, and Macintosh. In SAS, the following platforms are considered little endian: Intel ABI, Linux, OpenVMS Alpha, OpenVMS Integrity, Solaris on x64, Tru64 UNIX, and Windows.

**How the Bytes Are Ordered**

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

If an integer is negative, the “two's complement” representation is used. The high-order bit of the most significant byte of the integer is set on. For example, –2 would be represented in one, two, and four bytes on big endian platforms as FE, FF FE, and FF FF FF FE respectively. On little endian platforms, the representation would be FE, FF FF, and FE FF FF FF. These representations result from the output of the integer binary value –2 expressed in hexadecimal representation.
**Reading Data Generated on Big Endian or Little Endian Platforms**

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

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

<table>
<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.
<table>
<thead>
<tr>
<th>Language</th>
<th>2 Bytes or 8-Bit Systems</th>
<th>4 Bytes or 16-Bit Systems</th>
<th>8 Bytes or 64-Bit Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>short</td>
<td>int</td>
<td>long *</td>
</tr>
<tr>
<td>Java</td>
<td>short</td>
<td>int</td>
<td>long *</td>
</tr>
<tr>
<td>Visual Basic 6.0</td>
<td>short</td>
<td>long*</td>
<td>none</td>
</tr>
<tr>
<td>Visual Basic.NET</td>
<td>short</td>
<td>integer</td>
<td>long *</td>
</tr>
<tr>
<td>PL/I</td>
<td>fixed bin(15)</td>
<td>fixed bin(31)</td>
<td>fixed bin(63)</td>
</tr>
<tr>
<td>Fortran</td>
<td>integer*2</td>
<td>integer*4</td>
<td>integer*8</td>
</tr>
<tr>
<td>COBOL</td>
<td>comp pic 9(4)</td>
<td>comp pic 9(8)</td>
<td>comp pic 9(16)</td>
</tr>
<tr>
<td>IBM assembler</td>
<td>H</td>
<td>F</td>
<td>FD</td>
</tr>
</tbody>
</table>

* The size of integers declared as long depends on the operating environment.

---

**Working with Packed Decimal and Zoned Decimal Data**

**Definitions**

Packed decimal

specifies a method of encoding decimal numbers by using each byte to represent two decimal digits. Packed decimal representation stores decimal data with exact precision. The fractional part of the number is determined by the informat or format because there is no separate mantissa and exponent.

An advantage of using packed decimal data is that exact precision can be maintained. However, computations involving decimal data might become inexact due to the lack of native instructions.

Zoned decimal

specifies a method of encoding decimal numbers in which each digit requires one byte of storage. The last byte contains the number's sign as well as the last digit. Zoned decimal data produces a printable representation.

Nibble

specifies 1/2 of a byte.
Types of Data

Packed Decimal Data
A packed decimal representation stores decimal digits in each “nibble” of a byte. Each byte has two nibbles, and each nibble is indicated by a hexadecimal character. For example, the value 15 is stored in two nibbles, using the hexadecimal characters 1 and 5.

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

The following applies to packed decimal data representation:

• You can use the S370FPD format on all platforms to obtain the IBM mainframe configuration.

• You can have unsigned packed data with no sign indicator. The packed decimal format and informat handles the representation. It is consistent between ASCII and EBCDIC platforms.

• Note that the S370FPDU format and informat expects to have an F in the last nibble, whereas packed decimal expects no sign nibble.

Zoned Decimal Data
The following applies to zoned decimal data representation:

• A zoned decimal representation stores a decimal digit in the low order nibble of each byte. For all but the byte containing the sign, the high-order nibble is the numeric zone nibble (F on EBCDIC and 3 on ASCII).

• The sign can be merged into a byte with a digit, or it can be separate, depending on the representation. But the standard zoned decimal format and informat expects the sign to be merged into the last byte.

• The EBCDIC and ASCII zoned decimal formats produce the same printable representation of numbers. There are two nibbles per byte, each indicated by a hexadecimal character. For example, the value 15 is stored in two bytes. The first byte contains the hexadecimal value F1 and the second byte contains the hexadecimal value C5.

Packed Julian Dates
The following applies to packed Julian dates:

• The two formats and informats that handle Julian dates in packed decimal representation are PDJULI and PDJULG. PDJULI uses the IBM mainframe year computation, 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, whereas others must use software to emulate the computations. For example, the IBM mainframe has an Add Pack instruction to add packed decimal data, but the platforms that are based on Intel have no such instruction and must convert the decimal data into some other format.

**Languages Supporting Packed Decimal and Zoned Decimal Data**

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

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

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

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

<table>
<thead>
<tr>
<th>offset (in hex)</th>
<th>value</th>
<th>inst</th>
<th>label</th>
<th>directive</th>
</tr>
</thead>
<tbody>
<tr>
<td>+000000 00001C</td>
<td>00001C</td>
<td>2</td>
<td>PEX1</td>
<td>DC PL3'1'</td>
</tr>
<tr>
<td>+000003 0001D</td>
<td>0001D</td>
<td>3</td>
<td>PEX2</td>
<td>DC PL3'-1'</td>
</tr>
<tr>
<td>+000006 F0F0C1</td>
<td>ZEX1</td>
<td>4</td>
<td>DC ZL3'1'</td>
<td></td>
</tr>
<tr>
<td>+000009 F0F0D1</td>
<td>ZEX2</td>
<td>5</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 Informat

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

<table>
<thead>
<tr>
<th>Format</th>
<th>Data Type Representation</th>
<th>Corresponding Informat</th>
<th>Comments</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</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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>Comments</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: mmsstttF</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 0-based year from 1900</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>SMFSTAMP</td>
<td>Input layout is: xxxxxxxxxyyydddF, where yyyydddF is the packed Julian date; yyyy is a 0-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 0-based year from 1900</td>
</tr>
</tbody>
</table>

### Reading Dates and Times By Using the ISO 860 Basic and Extended Notations

#### ISO 8601 Formatting Symbols

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

- **n** specifies a number that represents the number of years, months, or days
- **P** indicates that the duration that follows is specified by the number of years, months, days, hours, minutes, and seconds
- **T** indicates that a time value follows. Any value with a time must begin with T.
Requirement  Time values that are read by the extended notation informats that begin with the characters E8601 must use an uppercase T.

\[ W \]
indicates that the duration is specified in weeks.

\[ Z \]
indicates that the time value is the time in Greenwich, England, or UTC time.

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

\[ yyyy \]
specifies a four-digit year

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

\[ dd \]
specifies a two-digit day, 01–1

\[ hh \]
specifies a two-digit hour, 00–24

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

\[ ss \]
specifies a two-digit second, 00–59

\[ fff \] \[ ffffffff \]
specifies an optional fraction of a second using the digits 0–9:

\[ fff \]
use 1 - 3 digits for values read by the $N8601B informat and the $N8601E informat

\[ ffffffff \]
use 1 - 6 digits for informat other than the $N8601B informat and the $N8601E informat

\[ Y \]
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><strong>Basic Notations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>YYYYMMDD</td>
<td>20130915</td>
<td>B8601DAw.</td>
</tr>
<tr>
<td>Time with no time zone offset</td>
<td>hhmmssnnnnnnn</td>
<td>155300322348</td>
<td>B8601TMw.d</td>
</tr>
<tr>
<td>Time with a time zone offset</td>
<td>hhmmss+</td>
<td>–hhmm</td>
<td>155300–0500</td>
</tr>
<tr>
<td></td>
<td>hhmmssZ</td>
<td>155300Z</td>
<td>B8601TZw.d</td>
</tr>
<tr>
<td>Convert to local time with a time zone offset</td>
<td>hhmmss+</td>
<td>–hhmm</td>
<td>155300+0500</td>
</tr>
<tr>
<td>Datetime with no time zone offset</td>
<td>yyyyymmddThhmmssnnnnnnn</td>
<td>20130915T155300</td>
<td>B8601DTw.d</td>
</tr>
<tr>
<td>Datetime with a time zone offset</td>
<td>yyyyymmddThhmmss+</td>
<td>–hhmm</td>
<td>20130915T155300+0500</td>
</tr>
<tr>
<td></td>
<td>yyyyymmddThhmmssZ</td>
<td>20130915T155300Z</td>
<td>B8601DZw.d</td>
</tr>
<tr>
<td>Date from a datetime and no time zone offset</td>
<td>yyyyymmdd</td>
<td>20130915</td>
<td>B8601DNw.</td>
</tr>
<tr>
<td><strong>Extended Notations</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Date</td>
<td>yyyy-mm-dd</td>
<td>2013-09-15</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+</td>
<td>–hh:mm</td>
<td>15:53:00+05:00</td>
</tr>
<tr>
<td>Convert to local time with a time zone offset</td>
<td>hh:mm:ss.nnnnnn+</td>
<td>–hh:mm</td>
<td>15:53:00+05:00</td>
</tr>
<tr>
<td>Datetime with no time zone offset</td>
<td>yyyy-mm-ddThh:mm:ss.nnnnnn</td>
<td>2013-09-15T15:53:00</td>
<td>E8601DTw.d</td>
</tr>
<tr>
<td>Datetime with a time zone offset</td>
<td>yyyy-mm-ddThh:mm:ss.nnnnnn+</td>
<td>–hh:mm</td>
<td>2013-09-15T15:53:00+05:00</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 2013-09-15T15:53:00+05:00 using the E8601DZ informat, the datetime value 1694861580 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 2013–09–15T10:53:00+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 20.

**Reading ISO 8601 Duration, Interval, and Datetime Values**

**Informats That Read Duration, Interval, and Datetime Values**
SAS uses two informats that read ISO datetime, duration, and interval values.

$N8601B informat

reads duration, interval, and datetime values that are specified in either the basic notation or the extended notation

$N8601E informat

reads duration, interval, and datetime values that are specified only in the extended notation

Use the $N8601E informat when you want to make sure that you are in compliance with the extended notation.

The datetime values that are read by these informats result in a SAS character representation. If you want a datetime value to be read as a numeric value, use the B8601DT informat, the B8601DZ informat, the E8601DT informat, or the E8601DZ informat.

**Complete Duration, Interval, and Datetime Notations**
The following table shows the formatting of duration, datetime, and interval values that can be read in the complete form:

<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>YYYY-MM-DDThhmmss</td>
<td>P20120915T155300</td>
</tr>
<tr>
<td></td>
<td>–YYYY-MM-DDThhmmss</td>
<td>–P20080915T155300</td>
</tr>
<tr>
<td>Duration - Extended Notation</td>
<td>YYYY-MM-DDThh:mm:ss</td>
<td>P2012-09-15T15:53:00</td>
</tr>
</tbody>
</table>
### Time Component

<table>
<thead>
<tr>
<th>ISO 8601 Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>–PYYYY-MM-DDThh:mm:ss</td>
<td>–P2012-09-15T15:53:00</td>
</tr>
</tbody>
</table>

### Duration - Basic and Extended Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>PnYnMnDTnHnMnS</td>
<td>P2y10m14dT20h13m45s</td>
</tr>
<tr>
<td>–PnYnMnDTnHnMnS</td>
<td>–P2n10m14dT20h13m45s</td>
</tr>
</tbody>
</table>

- **PnW (weeks)**: P6w

### Interval - Basic Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYYMMDDThhmmss/</td>
<td>20120915T155300/20141113T000000</td>
</tr>
<tr>
<td>YYYYMMDDThhmmss</td>
<td>P2y10M14dT20h13m45s/20120915T155300</td>
</tr>
<tr>
<td>YYYYMMDDThhmmss/</td>
<td>20120915T155300/P2y10M14dT20h13m45s</td>
</tr>
</tbody>
</table>

### Interval - Extended Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY-MM-DDThh:mm:ss/</td>
<td>2012-09-15T15:53:00/2014-11-13T00:00:00</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss</td>
<td>P2y10M14dT20h13m45s/2012-09-15T15:53:00</td>
</tr>
<tr>
<td>YYYY-MM-DDThh:mm:ss/</td>
<td>2012-09-15T15:53:00/P2y10M14dT20h13m45s</td>
</tr>
</tbody>
</table>

### Datetime - Basic Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYYMMDDThhmmss.fff+</td>
<td>20120915T155300</td>
</tr>
<tr>
<td>–hhmm</td>
<td></td>
</tr>
</tbody>
</table>

### Datetime - Extended Notation

<table>
<thead>
<tr>
<th>Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY-MM-DDThh:mm:ss.fff+</td>
<td>2012-09-15T15:53:00+04:30</td>
</tr>
<tr>
<td>–hhmm</td>
<td></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 using the `N8601B` informat or the `N8601E` informat, and the omitted component must be represented by a hyphen (-).

The following examples show duration, datetime, and interval values with omitted components:

- **p0003 -02 T10:31:33**  
  The omitted component is the number of days.

- **–p0003 -02 T -:31:33**  
  The omitted component is the number of hours.
x-09-15T15:x:x
The omitted components are the number of years, minutes, and seconds.

2012-09-15T15:x:00/2010-09-15T15:x:00
The omitted components are the minutes.

When reading values that contain a time zone offset, omitted components are not allowed. Use 00 in place of omitted components.

**Truncated Values**
SAS reads truncated duration, datetime, and interval values, where one or more lower order components is truncated because the value is 0 or the value is not significant.

The following list shows examples of truncated values:

- p00030202T1031
- 2012-09-15T15/2014-09-15T15:53
- –p0003-03-03T--:-:-
- P2y3m4dT5h6m
- 2012-09-xtx:x:x
- 2012

When reading values that contain a time zone offset, truncation is not allowed. Use 00 in place of truncated values.

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

<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>p0003-02-61T15:61:61</td>
<td>P0003-04-01T16:02:01</td>
</tr>
<tr>
<td>pl3m</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 Durations, Datetime, and Interval Values

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

- 201209.5
- P2012-09-15T10.33
- 2012-09-15/P0003-03-03,333
Chapter 4

Dictionary of Informats

Informats Documented in Other Publications ........................................... 247
Informats by Category ................................................................. 247

Dictionary ................................................................. 254

$ASCIIw. Informat ................................................................. 254
$BASE64Xw. Informat ............................................................... 255
$BINARYw. Informat ................................................................. 256
$CBw. Informat ................................................................. 257
$CHARw. Informat ................................................................. 258
$CHARZBw. Informat ................................................................. 260
$EBCDICw. Informat ................................................................. 261
$HEXw. Informat ................................................................. 262
$N8601Bw.d Informat ................................................................. 263
$N8601Ew.d Informat ................................................................. 265
$OCTALw. Informat ................................................................. 267
$PHEXw. Informat ................................................................. 268
$QUOTEw. Informat ................................................................. 269
$UPCASEw. Informat ................................................................. 270
$VARYINGw. Informat ................................................................. 272
$w. Informat ................................................................. 273
ANYYDTDTEw. Informat ................................................................. 273
ANYYDTDTMw. Informat ................................................................. 276
ANYYDTTMEw. Informat ................................................................. 279
B8601Clw.d Informat ................................................................. 282
B8601DAw. Informat ................................................................. 283
B8601DJw.d Informat ................................................................. 284
B8601DNw. Informat ................................................................. 286
B8601DTw.d Informat ................................................................. 287
B8601DXw. Informat ................................................................. 288
B8601DZw.d Informat ................................................................. 289
B8601LXw. Informat ................................................................. 290
B8601TMw.d Informat ................................................................. 291
B8601TXw. Informat ................................................................. 292
B8601TZw.d Informat ................................................................. 292
BINARYWw.d Informat ................................................................. 294
BITSw.d Informat ................................................................. 295
BZw.d Informat ................................................................. 296
CBw.d Informat ................................................................. 297
COMMAw.d Informat ................................................................. 298
COMMAXw.d Informat ................................................................. 299
DATEw. Informat ................................................................. 300
DATETIMEw. Informat .................................................. 301
DDMMYYw. Informat .................................................. 303
E8601DAw. Informat .................................................. 305
E8601DNw. Informat .................................................. 306
E8601DTw.d Informat .................................................. 307
E8601DXw. Informat .................................................. 308
E8601DZw.d Informat .................................................. 309
E8601LXw. Informat .................................................. 310
E8601LZw.d Informat .................................................. 311
E8601TMw.d Informat .................................................. 312
E8601TXw. Informat .................................................. 314
E8601TZw.d Informat .................................................. 314
FLOATw.d Informat .................................................. 316
HETw. Informat .................................................. 317
HHMSSw. Informat .................................................. 318
IBw.d Informat .................................................. 320
IBRw.d Informat .................................................. 321
IEEEw.d Informat .................................................. 322
JULIANw. Informat .................................................. 324
MDYAMPw.d Informat .................................................. 325
MMDDYYw. Informat .................................................. 327
MONEYw. Informat .................................................. 328
MSECw. Informat .................................................. 330
NUMXw.d Informat .................................................. 331
OCTALw.d Informat .................................................. 332
PDw.d Informat .................................................. 333
PDJULGw. Informat .................................................. 334
PDJULw. Informat .................................................. 336
PDTIMEw. Informat .................................................. 337
PERCENTw.d Informat .................................................. 338
PIBrw.d Informat .................................................. 339
PIBRw.d Informat .................................................. 341
PKw.d Informat .................................................. 342
PUNCH.d Informat .................................................. 343
RBw.d Informat .................................................. 344
RMFDURw. Informat .................................................. 346
RMFSTAMPw. Informat .................................................. 347
ROWw.d Informat .................................................. 348
S370FFw.d Informat .................................................. 350
S370FIBw.d Informat .................................................. 351
S370FIBUw.d Informat .................................................. 352
S370FPDw.d Informat .................................................. 354
S370FPDUw.d Informat .................................................. 355
S370FPIBw.d Informat .................................................. 356
S370FRBw.d Informat .................................................. 358
S370FZDw.d Informat .................................................. 359
S370FZDBw.d Informat .................................................. 360
S370FZDLw.d Informat .................................................. 361
S370FZDSw.d Informat .................................................. 363
S370FZDTw.d Informat .................................................. 364
S370FZDUw.d Informat .................................................. 365
SHRSTAMPw. Informat .................................................. 366
SMFSTAMPw. Informat .................................................. 367
STIMERw. Informat .................................................. 368
TIMEw. Informat .................................................. 369
TODSTAMPw. Informat .................................................. 371
Informats Documented in Other Publications


Informats by Category

There are five categories of informats in this list:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td>instructs SAS to read character data values into character variables.</td>
</tr>
<tr>
<td>Column Binary</td>
<td>instructs SAS to read data stored in column-binary or multipunched form into character 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” 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>Character</td>
<td>$ASCIIw. Informat (p. 254)</td>
<td>Converts ASCII character data to native format.</td>
</tr>
<tr>
<td></td>
<td>$BASE64Xw. Informat (p. 255)</td>
<td>Converts ASCII text into character data by using Base 64 encoding.</td>
</tr>
<tr>
<td></td>
<td>$SBINARYw. Informat (p. 256)</td>
<td>Converts binary data to character data.</td>
</tr>
<tr>
<td></td>
<td>$CHARw. Informat (p. 258)</td>
<td>Reads character data with blanks.</td>
</tr>
<tr>
<td></td>
<td>$CHARZBw. Informat (p. 260)</td>
<td>Converts binary 0s to blanks.</td>
</tr>
<tr>
<td></td>
<td>$SEBCDICw. Informat (p. 261)</td>
<td>Converts EBCDIC character data to native format.</td>
</tr>
<tr>
<td></td>
<td>$SHEXw. Informat (p. 262)</td>
<td>Converts hexadecimal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$SOCTALw. Informat (p. 267)</td>
<td>Converts octal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$SPHEXw. Informat (p. 268)</td>
<td>Converts packed hexadecimal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$QUOTEw. Informat (p. 269)</td>
<td>Removes matching quotation marks from character data.</td>
</tr>
<tr>
<td></td>
<td>$SUPCASEw. Informat (p. 270)</td>
<td>Converts character data to uppercase.</td>
</tr>
<tr>
<td></td>
<td>$VARYINGw. Informat (p. 270)</td>
<td>Reads character data of varying length.</td>
</tr>
<tr>
<td></td>
<td>$bw. Informat (p. 272)</td>
<td>Reads standard character data.</td>
</tr>
<tr>
<td>Column Binary</td>
<td>$SCBw. Informat (p. 257)</td>
<td>Reads standard character data from column-binary files.</td>
</tr>
<tr>
<td></td>
<td>CBw.d Informat (p. 297)</td>
<td>Reads standard numeric values from column-binary files.</td>
</tr>
<tr>
<td></td>
<td>PUNCH.d Informat (p. 343)</td>
<td>Reads whether a row of column-binary data is punched.</td>
</tr>
<tr>
<td></td>
<td>ROWw.d Informat (p. 348)</td>
<td>Reads a column-binary field down a card column.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>$SN8601Bw.d Informat (p. 263)</td>
<td>Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notations.</td>
</tr>
<tr>
<td></td>
<td>$SN8601Ew.d Informat (p. 265)</td>
<td>Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.</td>
</tr>
<tr>
<td></td>
<td>ANYDTDTEw. Informat (p. 273)</td>
<td>Reads and extracts the date value from various date, time, and datetime forms.</td>
</tr>
<tr>
<td></td>
<td>ANYDTDTMw. Informat (p. 276)</td>
<td>Reads and extracts datetime values from various date, time, and datetime forms.</td>
</tr>
<tr>
<td></td>
<td>ANYDTTMMw. Informat (p. 279)</td>
<td>Reads and extracts time values from various date, time, and datetime forms.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>B8601Clw.d Informat (p. 282)</td>
<td></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>B8601Da w. Informat (p. 283)</td>
<td></td>
<td>Reads date values that are specified using the ISO 8601 base notation yyyyymmdd.</td>
</tr>
<tr>
<td>B8601Djw.d Informat (p. 284)</td>
<td></td>
<td>Reads a Java date and time value that is in the form yyyyymmddhhmmss&lt;fff&gt;.</td>
</tr>
<tr>
<td>B8601Dn w. Informat (p. 286)</td>
<td></td>
<td>Reads date values that are specified using the ISO 8601 basic notation yyyyymmmd and returns SAS datetime values where the time portion of the value is 000000.</td>
</tr>
<tr>
<td>B8601Dt w.d Informat (p. 287)</td>
<td></td>
<td>Reads datetime values that are specified using the ISO 8601 basic notation yyyyymmddThhmmss&lt;fff&gt;.</td>
</tr>
<tr>
<td>B8601Dz w.d Informat (p. 289)</td>
<td></td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation yyyymmddThhmmss+–hhmm or yyyymmddThhmmss&lt;fff&gt;Z.</td>
</tr>
<tr>
<td>B8601Tm w.d Informat (p. 291)</td>
<td></td>
<td>Reads time values that are specified using the ISO 8601 basic notation hhmmss&lt;fff&gt;.</td>
</tr>
<tr>
<td>B8601Tz w.d Informat (p. 292)</td>
<td></td>
<td>Reads time values that are specified using the ISO 8601 basic time notation hhmmss&lt;ffff&gt;+–hhmm or hhmmss&lt;ffffff&gt;Z.</td>
</tr>
<tr>
<td>DATEw. Informat (p. 300)</td>
<td></td>
<td>Reads date values in the form ddmmyy or ddmmyyyy.</td>
</tr>
<tr>
<td>DATETIMEw. Informat (p. 301)</td>
<td></td>
<td>Reads datetime values in the form ddmmyy hh:mm:ss.ss or ddmmyyyyy hh:mm:ss.ss.</td>
</tr>
<tr>
<td>DDMMYYw. Informat (p. 303)</td>
<td></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 2 or 4 digits.</td>
</tr>
<tr>
<td>E8601Da w. Informat (p. 305)</td>
<td></td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601Dn w. Informat (p. 306)</td>
<td></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>E8601Dt w.d Informat (p. 307)</td>
<td></td>
<td>Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.&lt;fff&gt;.</td>
</tr>
<tr>
<td>E8601Dz w.d Informat (p. 309)</td>
<td></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;fff&gt; or yyyy-mm-ddThh:mm:ss.&lt;fff&gt;Z.</td>
</tr>
</tbody>
</table>
| E8601Lz w.d Informat (p. 311) | | Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation hh:mm:ss+–.
<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>hh:mm.&lt;ffffff&gt; or hh:mm:ss.&lt;ffffff&gt;Z and converts the values to the local time.</td>
</tr>
<tr>
<td>E8601TMw.d Informat (p. 312)</td>
<td></td>
<td>Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>E8601TZw.d Informat (p. 314)</td>
<td></td>
<td>Reads time values that are specified using the ISO 8601 extended time notation hh:mm:ss+</td>
</tr>
<tr>
<td>HHMMSSw. Informat (p. 318)</td>
<td></td>
<td>Reads hours, minutes, and seconds in the form hh:mm:ss or hhmmss.</td>
</tr>
<tr>
<td>JULIANw. Informat (p. 324)</td>
<td></td>
<td>Reads Julian dates in the form yyddd or yyyyddd.</td>
</tr>
<tr>
<td>MDYAMPMw.d Informat (p. 325)</td>
<td></td>
<td>Reads datetime values in the form mm-dd-yyyy hh:mm:ss.ss AM</td>
</tr>
<tr>
<td>MMDDYYw. Informat (p. 327)</td>
<td></td>
<td>Reads date values in the form mmddyy or mmddyyyy.</td>
</tr>
<tr>
<td>MONYYw. Informat (p. 328)</td>
<td></td>
<td>Reads month and year date values in the form mmyyy or mmyyyy.</td>
</tr>
<tr>
<td>MSECw. Informat (p. 330)</td>
<td></td>
<td>Reads TIME MIC values.</td>
</tr>
<tr>
<td>PDJULGw. Informat (p. 334)</td>
<td></td>
<td>Reads packed Julian date values in the hexadecimal form yyyydddF for IBM.</td>
</tr>
<tr>
<td>PDJULIw. Informat (p. 336)</td>
<td></td>
<td>Reads packed Julian dates in the hexadecimal format cyydddF for IBM.</td>
</tr>
<tr>
<td>PDTIMEw. Informat (p. 337)</td>
<td></td>
<td>Reads packed decimal time of SMF and RMF records.</td>
</tr>
<tr>
<td>RMFDURw. Informat (p. 346)</td>
<td></td>
<td>Reads duration intervals of RMF records.</td>
</tr>
<tr>
<td>RMFSTAMPw. Informat (p. 347)</td>
<td></td>
<td>Reads time and date fields of RMF records.</td>
</tr>
<tr>
<td>SHRSTAMPw. Informat (p. 366)</td>
<td></td>
<td>Reads date and time values of SHR records.</td>
</tr>
<tr>
<td>SMFSTAMPw. Informat (p. 367)</td>
<td></td>
<td>Reads time and date values of SMF records.</td>
</tr>
<tr>
<td>STIMERw. Informat (p. 368)</td>
<td></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>TIMEw. Informat (p. 369)</td>
<td></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>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>EOF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TODSTAMPw. Informat (p. 371)</td>
<td>Reads an eight-byte time-of-day stamp.</td>
<td></td>
</tr>
<tr>
<td>TUw. Informat (p. 373)</td>
<td>Reads timer units.</td>
<td></td>
</tr>
<tr>
<td>WEEKUw. Informat (p. 377)</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>
<td></td>
</tr>
<tr>
<td>WEEKVw. Informat (p. 379)</td>
<td>Reads a value in the form a week-number within a year and returns a SAS date value using the V algorithm.</td>
<td></td>
</tr>
<tr>
<td>WEEKWw. Informat (p. 381)</td>
<td>Reads a value in the form of a week-number within the year and returns a SAS date value using the W algorithm.</td>
<td></td>
</tr>
<tr>
<td>YMDDTTMw.d Informat (p. 383)</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 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>YYMMDDw. Informat (p. 385)</td>
<td>Reads date values in the form yymmdd or yyyyymmdd.</td>
<td></td>
</tr>
<tr>
<td>YYMMNNw. Informat (p. 387)</td>
<td>Reads date values in the form yyyymm or yymm.</td>
<td></td>
</tr>
<tr>
<td>YYQw. Informat (p. 388)</td>
<td>Reads quarters of the year in the form yyQq or yyyyQq.</td>
<td></td>
</tr>
<tr>
<td>ISO 8601</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SN8601Bw.d Informat (p. 263)</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 notations.</td>
<td></td>
</tr>
<tr>
<td>$N8601Bw.d Informat (p. 265)</td>
<td>Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.</td>
<td></td>
</tr>
<tr>
<td>B8601CIw.d Informat (p. 282)</td>
<td>Reads an IBM date and time value that includes a century marker, in the form cyyymmddhhmmss&lt;fff&gt;.</td>
<td></td>
</tr>
<tr>
<td>B8601DAw. Informat (p. 283)</td>
<td>Reads date values that are specified using the ISO 8601 base notation yymmddmm.</td>
<td></td>
</tr>
<tr>
<td>B8601DJw.d Informat (p. 284)</td>
<td>Reads a Java date and time value that is in the form yyyymmddhhmmss&lt;fff&gt;.</td>
<td></td>
</tr>
<tr>
<td>B8601DNw. Informat (p. 286)</td>
<td>Reads date values that are specified using the ISO 8601 base notation yyyymmdd and returns SAS datetime values where the time portion of the value is 000000.</td>
<td></td>
</tr>
<tr>
<td>B8601DTw.d Informat (p. 287)</td>
<td>Reads datetime values that are specified using the ISO 8601 basic notation yyyymmddThhmmss&lt;fff&gt;.</td>
<td></td>
</tr>
<tr>
<td>B8601DZw.d Informat (p. 289)</td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation yyyymmddThhmmss+</td>
<td>–hhmm or yyyymmddThhmmss&lt;fff&gt;Z.</td>
</tr>
<tr>
<td>B8601TMw.d Informat (p. 291)</td>
<td>Reads time values that are specified using the ISO 8601 basic notation hhmmss&lt;fff&gt;.</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>B8601TZw.d Informat (p. 292)</td>
<td>Reads time values that are specified using the ISO 8601 basic time notation hhmmss&lt;ff&gt;±</td>
</tr>
<tr>
<td></td>
<td>E8601DAw. Informat (p. 305)</td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td></td>
<td>E8601DNw. Informat (p. 306)</td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd and returns SAS datetime values where the time portion of the value is 000000.</td>
</tr>
<tr>
<td></td>
<td>E8601DTw.d Informat (p. 307)</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></td>
<td>E8601DZw.d Informat (p. 309)</td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime extended notation yyyy-mm-ddThh:mm:ss+</td>
</tr>
<tr>
<td></td>
<td>E8601LZw.d Informat (p. 311)</td>
<td>Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation hh:mm:ss+</td>
</tr>
<tr>
<td></td>
<td>E8601TMw.d Informat (p. 312)</td>
<td>Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.&lt;ffffff&gt;.</td>
</tr>
<tr>
<td></td>
<td>E8601TZw.d Informat (p. 314)</td>
<td>Reads time values that are specified using the ISO 8601 extended time notation hh:mm:ss+</td>
</tr>
<tr>
<td>Numeric</td>
<td>BINARYw.d Informat (p. 294)</td>
<td>Converts positive binary values to integers.</td>
</tr>
<tr>
<td></td>
<td>BISTSw.d Informat (p. 295)</td>
<td>Extracts bits.</td>
</tr>
<tr>
<td></td>
<td>BZW.d Informat (p. 296)</td>
<td>Converts blanks to 0s.</td>
</tr>
<tr>
<td></td>
<td>COMMAw.d Informat (p. 298)</td>
<td>Removes embedded characters.</td>
</tr>
<tr>
<td></td>
<td>COMMAXw.d Informat (p. 299)</td>
<td>Removes embedded periods, blanks, dollar signs, percent signs, hyphens, and closing parenthesis from the input data. An open parenthesis at the beginning of a field is converted to a minus sign. The COMMAX informat reverses the roles of the decimal point and the comma.</td>
</tr>
<tr>
<td></td>
<td>FLOATw.d Informat (p. 316)</td>
<td>Reads a native single-precision, floating-point value and divides it by 10 raised to the dth power.</td>
</tr>
<tr>
<td></td>
<td>HEXw. Informat (p. 317)</td>
<td>Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values.</td>
</tr>
<tr>
<td></td>
<td>IBw.d Informat (p. 320)</td>
<td>Reads native integer binary (fixed-point) values, including negative values.</td>
</tr>
<tr>
<td></td>
<td>IBRw.d Informat (p. 321)</td>
<td>Reads integer binary (fixed-point) values in Intel and DEC formats.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IEEEw.d Informat (p. 322)</td>
<td></td>
<td>Reads an IEEE floating-point value and divides it by 10 raised to the d th power.</td>
</tr>
<tr>
<td>NUMXw.d Informat (p. 331)</td>
<td></td>
<td>Reads numeric values with a comma in place of the decimal point.</td>
</tr>
<tr>
<td>OCTALw.d Informat (p. 332)</td>
<td></td>
<td>Converts positive octal values to integers.</td>
</tr>
<tr>
<td>PDw.d Informat (p. 333)</td>
<td></td>
<td>Reads data that are stored in IBM packed decimal format.</td>
</tr>
<tr>
<td>PERCENTw.d Informat (p. 338)</td>
<td></td>
<td>Reads percentages as numeric values.</td>
</tr>
<tr>
<td>PIBw.d Informat (p. 339)</td>
<td></td>
<td>Reads positive integer (fixed-point) values.</td>
</tr>
<tr>
<td>PIBRw.d Informat (p. 341)</td>
<td></td>
<td>Reads positive integer binary (fixed-point) values in Intel and DEC formats.</td>
</tr>
<tr>
<td>PKw.d Informat (p. 342)</td>
<td></td>
<td>Reads unsigned packed decimal data.</td>
</tr>
<tr>
<td>RBw.d Informat (p. 344)</td>
<td></td>
<td>Reads numeric data that is stored in real binary (floating-point) notation.</td>
</tr>
<tr>
<td>S370FFw.d Informat (p. 350)</td>
<td></td>
<td>Reads EBCDIC numeric data.</td>
</tr>
<tr>
<td>S370FIBw.d Informat (p. 351)</td>
<td></td>
<td>Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FIBUw.d Informat (p. 352)</td>
<td></td>
<td>Reads unsigned integer binary (fixed-point) values in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPDw.d Informat (p. 354)</td>
<td></td>
<td>Reads packed data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPDUw.d Informat (p. 355)</td>
<td></td>
<td>Reads unsigned packed decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPIBw.d Informat (p. 356)</td>
<td></td>
<td>Reads positive integer binary (fixed-point) values in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FRBw.d Informat (p. 358)</td>
<td></td>
<td>Reads real binary (floating-point) data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDw.d Informat (p. 359)</td>
<td></td>
<td>Reads zoned decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDBw.d Informat (p. 360)</td>
<td></td>
<td>Reads zoned decimal data in which zeros have been left blank.</td>
</tr>
<tr>
<td>S370FZDLw.d Informat (p. 361)</td>
<td></td>
<td>Reads zoned decimal leading-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDSw.d Informat (p. 363)</td>
<td></td>
<td>Reads zoned decimal separate leading-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDTw.d Informat (p. 364)</td>
<td></td>
<td>Reads zoned decimal separate trailing-sign data in IBM mainframe format.</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>S370FZDUw.d Informat (p. 365)</td>
<td>Reads unsigned zoned decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td></td>
<td>TRAILSGNw. Informat (p. 372)</td>
<td>Reads a trailing plus (+) or minus (−) sign.</td>
</tr>
<tr>
<td></td>
<td>VAXRBw.d Informat (p. 374)</td>
<td>Reads real binary (floating-point) data in VMS format.</td>
</tr>
<tr>
<td></td>
<td>VMSZNw.d Informat (p. 374)</td>
<td>Reads VMS and MicroFocus COBOL zoned numeric data.</td>
</tr>
<tr>
<td></td>
<td>w.d Informat (p. 376)</td>
<td>Reads standard numeric data.</td>
</tr>
<tr>
<td></td>
<td>ZDw.d Informat (p. 389)</td>
<td>Reads zoned decimal data.</td>
</tr>
<tr>
<td></td>
<td>ZDBw.d Informat (p. 391)</td>
<td>Reads zoned decimal data in which zeros have been left blank.</td>
</tr>
<tr>
<td></td>
<td>ZDVw.d Informat (p. 392)</td>
<td>Reads and validates zoned decimal data.</td>
</tr>
</tbody>
</table>

## Dictionary

### $ASCIIw. Informat

Converts ASCII character data to native format.

**Category:** Character

**Syntax**

$ASCIIw.

**Syntax Description**

- **w**
  - Specifies the width of the input field.
  - **Default:** 1 if the length of the variable is undefined. Otherwise, the default is the length of the variable.
  - **Range:** 1–32767

**Details**

If ASCII is the native format, no conversion occurs.

**Comparisons**

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

### Example

```plaintext
input @1 name $ascii3.;
```

<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>818283</td>
</tr>
<tr>
<td>ABC</td>
<td>C1C2C3</td>
</tr>
<tr>
<td>(;)</td>
<td>4D5D5E</td>
</tr>
</tbody>
</table>

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

### $BASE64Xw$. Informat

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

- **Category:** Character
- **Alignment:** Left

#### Syntax

```plaintext
$BASE64Xw.
```

#### 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 /.

The following are some uses of Base 64 encoding:

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

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

Example

```plaintext
input @1 b64exmpl $base64x64.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>RkNBMDFBNzk5M0JD</td>
<td>FCA01A7993BC</td>
</tr>
<tr>
<td>TXIQYXNd29yZA==</td>
<td>MyPassword</td>
</tr>
<tr>
<td>d3d3Lm15ZG9tYWluLmNvbi9teWhpZGRJb1VSTA==</td>
<td><a href="http://www.mydomain.com/">www.mydomain.com/</a> myhiddenURL</td>
</tr>
</tbody>
</table>

See Also

• The XMLDOUBLE option of the “LIBNAME Statement Syntax” in SAS XML LIBNAME Engine: User's Guide

Formats:

• “$BASE64Xw. Format” on page 43

$BINARYw. Informat

Converts binary data to character data.

**Category:** Character

**Syntax**

$BINARYw.

**Syntax Description**

`w`

specifies the width of the input field. Because eight bits of binary information represent one character, every eight characters of input that $BINARYw. reads becomes one character value stored in a variable.

If `w` < 8, $BINARYw. reads the data as `w` characters followed by 0s. Thus, $BINARY4. reads the characters 0101 as 01010000, which converts to an EBCDIC & or an ASCII p. If `w` > 8 but is not a multiple of 8, $BINARYw. reads up to the largest multiple of 8 that is less than `w` before converting the data.

**Default** 8

**Range** 1–32767
Details

The $BINARY_w$. informat does not interpret actual binary data, but it converts a string of characters that contains only 0s or 1s as if it is actual binary information. Therefore, use only the character digits 1 and 0 in the input, with no embedded blanks. $BINARY_w$. ignores leading and trailing blanks.

To read representations of binary codes for unprintable characters, enter an ASCII or EBCDIC equivalent for a particular character as a string of 0s and 1s. The $BINARY_w$. informat converts the string to its equivalent character value.

Comparisons

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

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100110001001101</td>
<td>ASCII</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>LM</td>
<td>&lt;{</td>
<td></td>
</tr>
</tbody>
</table>

$CBw. Informat

Reads standard character data from column-binary files.

Category: Column Binary

Syntax

$CBw.$

Syntax Description

$w$

specifies the width of the input field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>1–32767</td>
</tr>
</tbody>
</table>

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 two 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>200A</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 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 297
- “PUNCH.d Informat” on page 343
- “ROWw.d Informat” on page 348

---

### $CHARw$. Informat

Reads character data with blanks.

**Category:** Character

**Syntax**

$CHARw$.

**Syntax Description**

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

**Default**

8 if the length of the variable is undefined. Otherwise, the 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, then by default SAS interprets any blank embedded within data as a field delimiter, including leading blanks.

Comparisons

• The $CHARw. informat is almost identical to the $w. informat. However, $CHARw. does not trim leading blanks or convert a single period in the input data field to a blank, while the $w. informat does.

• Use the table below to compare the SAS informat $CHAR8. with notation in other programming languages:

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

Example

    input @1 name $char5.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>XYZ##</td>
</tr>
<tr>
<td>XYZ</td>
<td>#XYZ#</td>
</tr>
<tr>
<td>.</td>
<td>##.###</td>
</tr>
<tr>
<td>X YZ</td>
<td>#X#YZ</td>
</tr>
</tbody>
</table>

* The character # represents a blank space.
$\text{CHARZB}w$. Informat

Converts binary 0s to blanks.

**Category:** Character

**Syntax**

$\text{CHARZB}w$.  

**Syntax Description**

$w$

specifies the width of the input field.

**Default**

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

**Range**

$1–32767$

**Details**

The $\text{CHARZB}w$. informat does not trim leading and trailing blanks in character data before it stores values.

**Comparisons**

The $\text{CHARZB}w$. informat is identical to the $\text{CHAR}w$. informat except that $\text{CHARZB}w$. converts any byte that contains a binary 0 to a blank character.

**Example**

```plaintext
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 one byte of binary data, and each byte corresponds to one character.

** The character # represents a blank space.
$EBCDIC$. Informat

Converts EBCDIC character data to native format.

**Category:** Character

## Syntax

$EBCDIC$.w.

## Syntax Description

w

specifies the width of the input field.

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

**Range** 1–32767

## Details

If EBCDIC is the native format, no conversion occurs.

**Note:** Any time a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option on either ASCII or EBCDIC environments. When that you read an EBCDIC text file on an ASCII platform, it is recommended that you specify the ENCODING= option in the FILENAME or INFILE statement. However, if you use the DSD and the DLM= or DLMSTR= options in the FILENAME or INFILE statement, the ENCODING= option is a requirement because these options require certain characters in the session encoding (for example, quotation marks, commas, and blanks). The use of encoding-specific informats should be reserved for use with true binary files. That is, they contain both character and non-character fields.

## Comparisons

- On an IBM mainframe system, $EBCDIC$.w. behaves like the $CHAR$.w. informat.
- On all other systems, $EBCDIC$.w. converts EBCDIC data to ASCII.

## Example

input @1 name $ebcdic3.

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result 1</th>
<th>Result 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------</td>
<td>ASCII</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>qrs</td>
<td>717273</td>
<td>9899A2</td>
</tr>
<tr>
<td>QRS</td>
<td>515253</td>
<td>D8D9E2</td>
</tr>
</tbody>
</table>
The results are hexadecimal representations of codes for characters. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one character value.

**$HEXw. Informat**

Converts hexadecimal data to character data.

**Category:** Character

**See:**
- "$HEXw. Informat: UNIX" in *SAS Companion for UNIX Environments*
- "$HEXw. Informat: Windows" in *SAS Companion for Windows*

**Syntax**

$HEXw.

**Syntax Description**

w

specifies the number of digits of hexadecimal data.

If w=1, $HEXw. pads a trailing hexadecimal 0. If w is an odd number that is greater than 1, then $HEXw. reads w–1 hexadecimal characters.

**Default** 2

**Range** 1–32767

**Details**

The $HEXw. informat converts every two digits of hexadecimal data into one byte of character data. Use $HEXw. 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 into one byte of numeric data.

**Example**

```sas
   input @1 name $hex4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2B3B3E</td>
<td>4E5E6E</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;+-&gt;</td>
<td>ASCII EBCDIC</td>
</tr>
</tbody>
</table>
| $C6C       | 11 '

# Chapter 4 • Dictionary of Informats
$N8601Bw.d$ Informat

Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notations.

- **Categories**: Date and Time
  - ISO 8601
- **Alignment**: Left
- **Restriction**: UTC time zone offset values are not supported.
- **Supports**: ISO 8601 Element 5.4.4, complete representation

### Syntax

$N8601Bw.d$

#### Syntax Description

- **$w$**
  - Specifies the width of the input field.
  - **Default**: 50
  - **Range**: 1–200
  - **Requirement**: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

- **$d$**
  - Specifies the number of digits to the right of the decimal point in the seconds value.
  - This argument is optional.
  - **Default**: 0
  - **Range**: 0–3

#### Details

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

<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>PyyyyymmddThhmmss</td>
<td>P00020304T050607</td>
</tr>
<tr>
<td></td>
<td>PnYnMnDTnHnMn.fjjS</td>
<td>P2y10m14dT20h13m45.222s</td>
</tr>
<tr>
<td></td>
<td>PnW</td>
<td>P6w</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.

**Comparisons**

The $N8601B informat reads durations, intervals, and datetimes that are specified in either the basic or extended notation.

The $N8601E informat reads durations, intervals, and datetimes that are specified only in the extended notation. Use the $N8601E informat when you need to ensure compliance with the extended notation.

**Example**

```
input @1 i860 $n8601b.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0002-04-05T5:1:12</td>
<td>0002405050112FFC</td>
</tr>
<tr>
<td>2012-09-15T15:53:00/2010-09-15T00:00:00</td>
<td>2012915155300FFD2010915000000FFD</td>
</tr>
<tr>
<td>0033-01-04T3:2:55/2012-09-15T15:53:00</td>
<td>0033104030255FFC2012915155300FFD</td>
</tr>
</tbody>
</table>

**See Also**

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237
## $N8601Ew.d Informat

Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.

### Categories:
Date and Time  
ISO 8601

### Alignment:
Left

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

### Supports:
ISO 8601 Element 5.4.4, complete representation

## Syntax

$N8601Ew.d$

### Syntax Description

\( w \)  

specifies the width of the input field.

**Default**  
50

**Range**  
1–200

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

\( d \)  

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

**Default**  
0

**Range**  
0–3

### Details

The $N8601E$ informat reads ISO 8601 duration, interval, and datetime values that can be specified in the following extended notations:

<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>PnW</td>
<td>P6w</td>
</tr>
<tr>
<td>Interval</td>
<td>yyyy-mm-ddThh:mm:ss.fff/ yyyy-mm-ddThh:mm:ss.fff</td>
<td>2012-09-15T15:53:00/2014-11-13T00:00</td>
</tr>
<tr>
<td></td>
<td>yyyy-mm-ddThh:mm:ss.fff</td>
<td>2012-09-15T15:53:00/ P2Y10M14DT20H13M45S</td>
</tr>
<tr>
<td></td>
<td>PnYnMnDTnHnMns.fff</td>
<td></td>
</tr>
</tbody>
</table>


Time Component | ISO 8601 Notation | Example
--- | --- | ---
Datetime | yyyy-mm-ddThh:mm:ss.fff | 2012-09-15T15:53:00

$n$

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

$P$

is the character that is used to indicate that the duration that follows is specified by the number of years, months, days, hours, minutes, and seconds.

$W$

is the character that is used to designate that the duration is specified in weeks.

$T$

is the character that is used to designate that a time value follows. If all time values are 0, $T$ is not required.

$/$

in an interval, is used to separate the beginning and ending datetime values.

$yyyy$

specifies a four-digit year.

$mm$

specifies a two-digit month between 01 and 12.

$dd$

specifies a two-digit day between 01 and 31.

$hh$

specifies a two-digit hour between 00 and 23.

$mm$

specifies a two-digit minute between 00 and 59.

$ss$

specifies a two-digit second between 00 and 59.

$fff$

specifies an optional fraction of a second with a precision of up to three digits, where each digit is between 0 and 9.

$Y$

is the character that is used to designate years in a duration.

$M$

is the character that is used to designate months in a duration.

$D$

is the character that is used to designate days in a duration.

$H$

is the character that is used to designate hours in a duration.

$M$

is the character that is used to designate minutes in a duration.

$S$

is the character that is used to designate seconds in a duration.
Comparisons

The $N8601E informat reads valid durations, intervals, and datetimes that are specified only in the extended notation.

The $N8601B informat reads valid durations, intervals, and datetimes that are specified in either the basic or extended notation.

Use the $N8601E informat when you need to ensure compliance with the extended notation.

Example

```latex
input @1 1860 $n8601e.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0002-04-05t5:1:12s</td>
<td>0002405050112FFC</td>
</tr>
<tr>
<td>2012-09-15T15:53:00/2014-09-15T00:00</td>
<td>2012915155300FFD2014915000000FFD</td>
</tr>
<tr>
<td>p0033-01-04T3:2:55/2012-09-15T15:53:00</td>
<td>0033104030255FFFC2012915155300FFD</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

---

$OCTAL_w. Informat

Converts octal data to character data.

**Category:** Character

**Syntax**

$OCTAL_w.$

**Syntax Description**

\(w\)

specifies the width of the input field in bits. Because one digit of octal data represents three bits of binary information, increment the value of \(w\) by three for every column of octal data that $OCTAL_w.$ 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 it to its equivalent character value.

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

**Comparisons**

The OCTALw. informat reads octal data and converts 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>ASCII</td>
</tr>
<tr>
<td></td>
<td>&lt;</td>
</tr>
<tr>
<td></td>
<td>L</td>
</tr>
</tbody>
</table>

**$PHEXw. Informat**

Converts packed hexadecimal data to character data.

**Category:** Character

**Syntax**

$PHEXw$.  

**Syntax Description**

w specifies the number of bytes in the input.

When you use $PHEXw$. to read packed hexadecimal data, the length of the variable is the number of bytes that are required to store the resulting character value, not w.

In general, a character variable whose length is implicitly defined with $PHEXw$. has a length of $2^w - 1$.

**Default** 2

**Range** 1–32767

**Details**

Packed hexadecimal data are like packed decimal data, except that all hexadecimal characters are valid. In packed hexadecimal data, the value of the low-order nibble has no meaning. In packed decimal data, the value of the low-order nibble indicates the sign
of the numeric value that the data represent. The $PHEXw.$ 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 them to numeric data.

**Example**

```
input @1 devaddr $phex2.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001111000001111</td>
<td>1E0</td>
</tr>
</tbody>
</table>

* The data line represents two 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.

**Category:** 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**

```
input @1 name $quote7.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>----+----</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SAS</td>
</tr>
<tr>
<td>'SAS'</td>
<td>SAS</td>
</tr>
<tr>
<td>&quot;SAS&quot;</td>
<td>SAS</td>
</tr>
</tbody>
</table>
### $UPCASE_w. Informat

Converts character data to uppercase.

**Category:** Character

**Syntax**

$UPCASE_w.$

**Syntax Description**

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

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

  **Range** 1–32767

**Details**

Special characters, such as hyphens, are not altered.

**Example**

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SAS's&quot;</td>
<td>SAS 's</td>
</tr>
</tbody>
</table>

### $VARYING_w. Informat

Reads character data of varying length.

**Category:** Character

**Syntax**

$VARYING_w. length-variable
Syntax Description

\( w \)

specifies the maximum width of a character field for all the records in an input file.

**Default**

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

**Range**

1–32767

\( length-variable \)

specifies a numeric variable that contains the width of the character field in the current record. SAS obtains the value of \( length-variable \) by reading it directly from a field that is described in an INPUT statement or by calculating its value in the DATA step.

**Restriction**

\( length-variable \) cannot be an array reference.

**Requirement**

You must specify \( length-variable \) immediately after $VARYINGw. in an INPUT statement.

**Tips**

If the value of \( length-variable \) is negative or missing, SAS reads no data from the corresponding record.

If the value of \( length-variable \) is 0, the value of the variable is a blank character. A value of 0 for \( length-variable \) enables you to read zero-length records and fields.

If a variable has been read using an informat other than the $VARYING. informat, and then the same data is read into the same variable that uses the $VARYING. informat where \( length-variable \) is 0, then the previous value is overwritten with a blank value.

If \( length-variable \) is greater than 0 but less than \( w \), SAS reads the number of columns that are specified by \( length-variable \). Then SAS pads the value with trailing blanks up to the maximum width that is assigned to the variable.

If \( length-variable \) is greater than or equal to \( w \), SAS reads \( w \) columns.

Details

Use $VARYINGw. when the length of a character value differs from record to record. After reading a data value with $VARYINGw., the pointer's position is set to the first column after the value.

Examples

**Example 1: Obtaining a Current Record Length Directly**

```
input fwidth 1. name $varying9. fwidth;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```
**Example 2: Obtaining a Record Length Indirectly**

Use the LENGTH= option in the INFILE statement to obtain a record length indirectly. The input data lines and results follow the explanation of the SAS statements.

```sas
data one;
  infile file-specification length=reclen;
  input @;
  fwidth=reclen-9;
  input name $ 1-9         @10 class $varying20. fwidth;
run;
```

The LENGTH= option in the INFILE statement assigns the internally stored record length to RECLLEN when the first INPUT statement executes. The trailing @ holds the record for another INPUT statement. Next, the assignment statement calculates the value of the varying-length field by subtracting the fixed-length portion of the record from the total record length. The variable FWIDTH contains the length of the last field and becomes the length-variable argument to the $VARYING20. informat.

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

**$w. Informat**

Reads standard character data.

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

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>XYZ</td>
<td>XYZ##</td>
</tr>
<tr>
<td>XYZ</td>
<td>XYZ##</td>
</tr>
<tr>
<td>.</td>
<td></td>
</tr>
<tr>
<td>X YZ</td>
<td>X#YZ#</td>
</tr>
</tbody>
</table>

* The character # represents a blank space.

ANYDTDTEw. Informat

Reads and extracts the date value from various date, time, and datetime forms.

Category: Date and Time

Syntax

```
ANYDTDTEw.
```
Syntax Description

\[ w \]

specifies the width of the input field.

Default \( 9 \)

Range \( 5–32 \)

Details

The ANYDTDTE informat reads input data that corresponds to any of the following informats or date, time, or datetime forms and extracts the date part from the derived value.

<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>01JAN12</td>
<td>MONYY</td>
<td>JAN12</td>
</tr>
<tr>
<td></td>
<td>01JAN2012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DATETIME</td>
<td>01JAN12 14:30:08</td>
<td>TIME</td>
<td>14:30</td>
</tr>
<tr>
<td></td>
<td>01JAN2012 14:30:08.5</td>
<td></td>
<td>14:30:08.05</td>
</tr>
<tr>
<td>DDMMYY</td>
<td>010112</td>
<td>YMDDTTM</td>
<td>12-01-01 11:23</td>
</tr>
<tr>
<td></td>
<td>01012012</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JULIAN</td>
<td>12001</td>
<td>YYMMDD</td>
<td>120101</td>
</tr>
<tr>
<td></td>
<td>2012001</td>
<td></td>
<td>20120101</td>
</tr>
<tr>
<td>MDYAMPM</td>
<td>01-01-12 3:53 pm</td>
<td>YYQ</td>
<td>12Q1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2012Q1</td>
</tr>
<tr>
<td>MMDDYY</td>
<td>010112</td>
<td>YY&lt;YY&gt;&lt;MM&gt;</td>
<td>12/01</td>
</tr>
<tr>
<td></td>
<td>01012012</td>
<td>x</td>
<td>2012-01</td>
</tr>
<tr>
<td>MMxYY&lt;YY&gt;^</td>
<td>01/12</td>
<td>month-day-year</td>
<td>January 1, 2012</td>
</tr>
<tr>
<td></td>
<td>01-2012</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, then SAS assumes a date of 01JAN1960.

It is possible for input data such as 01-02-03 or 01-02 to be ambiguous with respect to the month, day, and year. In this case, the DATESTYLE system option indicates the order of the month, day, and year.

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 zero.
Comparisons

The ANYDTDTE informat extracts the date part from the derived value. The ANYDTDTM informat extracts the datetime part. The ANYDTTME informat extracts the time part.

Example

    input dateinfo anytdtde21.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Informat Form</th>
<th>Result</th>
<th>Formatted with the DATEw. Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>01JAN12</td>
<td>DATE</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
<tr>
<td>01JAN2012</td>
<td>DATETIME</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
<tr>
<td>01/01/2012</td>
<td>DDMMYY</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
<tr>
<td>2012001</td>
<td>JULIAN</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
<tr>
<td>01/01/12</td>
<td>MMDDYY</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
<tr>
<td>JAN2012</td>
<td>MONYY</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
<tr>
<td>14:30</td>
<td>TIME</td>
<td>0</td>
<td>01JAN60</td>
</tr>
<tr>
<td>20120101</td>
<td>YYMMDD</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
<tr>
<td>12q1</td>
<td>YYQ</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
<tr>
<td>January 1, 2012</td>
<td>none</td>
<td>18993</td>
<td>01JAN12</td>
</tr>
</tbody>
</table>

See Also

Informats:
- “ANYDTDTMw. Informat” on page 276
- “ANYDTTMEw. Informat” on page 279
- “DATEw. Informat” on page 300
- “DATETIMEw. Informat” on page 301
- “DDMMYYw. Informat” on page 303
- “JULIANw. Informat” on page 324
- “MDYAMPWd Informat” on page 325
- “MMDDYYw. Informat” on page 327
ANYDTDTMw. Informat

Reads and extracts datetime values from various date, time, and datetime forms.

**Category:** 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** 1–32

**Details**

The ANYDTDTM informat reads data that is in the form of any of the following informats or date/time forms, and extracts the datetime part from the derived value:

<table>
<thead>
<tr>
<th>Informat or Form of Input</th>
<th>Example Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>01JAN12</td>
</tr>
<tr>
<td></td>
<td>01JAN2012</td>
</tr>
<tr>
<td>DATETIME</td>
<td>01JAN12 14:30:08</td>
</tr>
<tr>
<td></td>
<td>01JAN2012 14:30:08.5</td>
</tr>
<tr>
<td>DDMM&lt;YY&gt;YY ** ***</td>
<td>010112</td>
</tr>
<tr>
<td></td>
<td>01012012</td>
</tr>
<tr>
<td>JULIAN</td>
<td>12001</td>
</tr>
<tr>
<td></td>
<td>2012001</td>
</tr>
<tr>
<td>Informat or Form of Input</td>
<td>Example Data</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>MMDD&lt;YY&gt;YY ** ***</td>
<td>010112</td>
</tr>
<tr>
<td></td>
<td>01012012</td>
</tr>
<tr>
<td>MMx&lt;YY&gt;YY * **</td>
<td>01/12</td>
</tr>
<tr>
<td></td>
<td>01-2012</td>
</tr>
<tr>
<td>MDYAMPM **</td>
<td>01/01/12 02:30:08 AM</td>
</tr>
<tr>
<td></td>
<td>01/01/2012 02:30:08 AM</td>
</tr>
<tr>
<td>MONYY</td>
<td>JAN12</td>
</tr>
<tr>
<td></td>
<td>JAN2012</td>
</tr>
<tr>
<td>TIME</td>
<td>14.30</td>
</tr>
<tr>
<td></td>
<td>14:30:08.05</td>
</tr>
<tr>
<td>&lt;YY&gt;YYMMDD **</td>
<td>120101</td>
</tr>
<tr>
<td></td>
<td>20120101</td>
</tr>
<tr>
<td>&lt;YY&gt;YYQ **</td>
<td>12Q1</td>
</tr>
<tr>
<td></td>
<td>2012Q1</td>
</tr>
<tr>
<td>&lt;YY&gt;YYxMM * **</td>
<td>12/01</td>
</tr>
<tr>
<td></td>
<td>2012/01</td>
</tr>
<tr>
<td>* x is a special character that separates the month from the year.</td>
<td></td>
</tr>
<tr>
<td>** &lt;YY&gt; indicates the century is optional.</td>
<td></td>
</tr>
<tr>
<td>*** IF the month and day values are ambiguous, the value for the DATESTYLE= system option is used to determine the order.</td>
<td></td>
</tr>
</tbody>
</table>

If the input value is a time-only value, then SAS assumes a date of 01JAN1960. If the input value is a date-only value, then SAS assumes a time of 12:00 midnight. Input time values must include hours and minutes. If any part of a date in the input value is missing in the input value, or if the hour and minutes in a time value are missing or out of range, then the value read is a SAS missing value.

The input values for the preceding informats are mutually exclusive except for MMDDYY, DDMMYY, or YYMMDD when two-digit years are used. It is possible for input data such as 01-02-03 or 01-02 to be ambiguous with respect to the month, day, and year. In this case, the DATESTYLE system option indicates the order of the month, day, and year.

The ANYDTTME informat uses the following rules when reading colons and periods in time values:
Use of Colons and Periods

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>a single colon in the value ( h:m ) indicates hours and minutes</td>
</tr>
<tr>
<td>14:30</td>
</tr>
<tr>
<td>two colons in the value ( h:m:s ) indicate hours, minutes, and seconds</td>
</tr>
<tr>
<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>
</tr>
<tr>
<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>
</tr>
<tr>
<td>12.25.2012</td>
</tr>
</tbody>
</table>

Comparisons

The ANYTDTE informat extracts the date part from the derived value. The ANYDTDTM informat extracts the datetime part. The ANYDTTME informat extracts the time part.

Example

```
input dateinfo anydt dtm21.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Informat or Form of Data</th>
<th>Result</th>
<th>Formatted with DATETIMEw.d Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>01JAN2012</td>
<td>DATE</td>
<td>1640995200</td>
<td>01JAN12:00:00:00</td>
</tr>
<tr>
<td>01JAN2012 14:30:08.5</td>
<td>DATETIME</td>
<td>1641047408.5</td>
<td>01JAN12:14:30:09</td>
</tr>
<tr>
<td>01012012</td>
<td>DDMYY</td>
<td>1640995200</td>
<td>01JAN12:00:00:00</td>
</tr>
<tr>
<td>2012001</td>
<td>JULIAN</td>
<td>1546387200</td>
<td>01JAN12:00:00:00</td>
</tr>
<tr>
<td>01/01/12</td>
<td>MMDDYY</td>
<td>1546387200</td>
<td>01JAN12:00:00:00</td>
</tr>
<tr>
<td>01-12</td>
<td>MMxYY</td>
<td>1546387200</td>
<td>01JAN12:00:00:00</td>
</tr>
<tr>
<td>JAN2012</td>
<td>MONYY</td>
<td>1546387200</td>
<td>01JAN12:00:00:00</td>
</tr>
<tr>
<td>14:30</td>
<td>TIME</td>
<td>52200</td>
<td>01JAN60:14:30:00</td>
</tr>
<tr>
<td>20120101</td>
<td>YYMMDD</td>
<td>1546387200</td>
<td>01JAN12:00:00:00</td>
</tr>
<tr>
<td>12Q1</td>
<td>YYQ</td>
<td>1546387200</td>
<td>01JAN12:00:00:00</td>
</tr>
</tbody>
</table>
ANYDTTMEw. Informat

Reads and extracts time values from various date, time, and datetime forms.

Syntax

\[ \text{ANYDTTME}w. \]

Syntax Description

\( w \)

specifies the width of the input field.

Default \( 8 \)

Range \( 1–32 \)

Details

The ANYDTTME informat reads input data that corresponds to any of the following informats or forms.
Informat or Form of
Input | Example Data | Informat or Form of
Input | Example Data
--- | --- | --- | ---
DATE | 01JAN12 | MONYY | JAN12
| 01JAN2012 |  |  |  
DATETIME | 01JAN12 14:30:08 | YYMMDD | 120101
| 01JAN2012 14:30:08.5 |  | 20120101 |
DDMMYY | 010112 | YYQ | 12Q1
| 01012012 |  | 2012Q1 |
JULIAN | 12001 | YYQ | 12Q1
| 2012001 |  | 2012Q1 |
MMDDYY | 010112 | month-day-year | January 1, 2012
| 01012012 |  | 2012-01 |

If the input value is a time-only value, then SAS assumes a date of 01JAN1960. If the input value is a date value only, then SAS assumes a time of 12:00 midnight.

It is possible for input data such as 01-02-03 or 01-02 to be ambiguous with respect to the month, day, and year. In this case, the DATESTYLE system option indicates the order of the month, day, and year.

The ANYDTTME informat uses the following rules when reading colons and periods in time values:

<table>
<thead>
<tr>
<th>Use of Colons and Periods</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>a single colon in the value \textit{h:m} indicates hours and minutes</td>
<td>14:30</td>
</tr>
<tr>
<td>two colons in the value \textit{h:m:s} indicate hours, minutes, and seconds</td>
<td>14:30:08</td>
</tr>
<tr>
<td>a single period in the value \textit{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>12.25.2012</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
input dateinfo anydttme21.;
```

<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>01JAN12</td>
<td>DATE</td>
<td>0</td>
<td>00:00:00</td>
</tr>
<tr>
<td>01JAN2012</td>
<td>DATETIME</td>
<td>52208.5</td>
<td>14:30:09</td>
</tr>
<tr>
<td>010112</td>
<td>DDMYY</td>
<td>0</td>
<td>00:00:00</td>
</tr>
<tr>
<td>2012001</td>
<td>JULIAN</td>
<td>0</td>
<td>00:00:00</td>
</tr>
<tr>
<td>01012012</td>
<td>MMDDYY</td>
<td>0</td>
<td>00:00:00</td>
</tr>
<tr>
<td>JAN2012</td>
<td>MONYY</td>
<td>0</td>
<td>00:00:00</td>
</tr>
<tr>
<td>14:30:08.5</td>
<td>TIME</td>
<td>52208.5</td>
<td>14:30:09</td>
</tr>
<tr>
<td>20120101</td>
<td>YYMMDD</td>
<td>0</td>
<td>00:00:00</td>
</tr>
<tr>
<td>12Q1</td>
<td>YYQ</td>
<td>0</td>
<td>00:00:00</td>
</tr>
<tr>
<td>January 1, 2012</td>
<td>month-day-year</td>
<td>0</td>
<td>00:00:00</td>
</tr>
</tbody>
</table>

See Also

**Informats:**

- “ANYDTDTEw. Informat” on page 273
- “ANYDTDTMw. Informat” on page 276
- “DATEw. Informat” on page 300
- “DATETIMEw. Informat” on page 301
- “DDMMYYw. Informat” on page 303
- “JULIANw. Informat” on page 324
- “MMDDYYw. Informat” on page 327
- “MONYYw. Informat” on page 328
- “TIMEw. Informat” on page 369
- “YYMMDDw. Informat” on page 385
- “YYQw. Informat” on page 388
B8601CIw.d Informat

Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss<fff>.

**Categories:** Date and Time  
ISO 8601  

**Alignment:** Left

### Syntax

\texttt{B8601CIw.d}

### Syntax Description

\texttt{w}

- Specifies the width of the input field.
- **Default:** 16
- **Range:** 10–26

\texttt{d}

- Specifies the number of digits to the right of the decimal point in the seconds value.
- **Default:** 0
- **Range:** 0–6

### Details

The B8601CI informat reads time values that are specified in the IBM time notation cyymmddhhmmss<fff>:

\texttt{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.

\texttt{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.

\texttt{yy}

- Is a two-digit year between 00 and 99.

\texttt{mm}

- Is a two-digit month (zero padded) between 01 and 12.
"dd"
is a two-digit day of the month (zero padded) between 01 and 31.

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

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

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

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

**Example**

```
input @1 bci b8601ci.;
```

<table>
<thead>
<tr>
<th>Date and Time</th>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>January 1, 1900 12:00:00</td>
<td>0000101120000</td>
<td>-1893326400</td>
</tr>
<tr>
<td>October 1, 2011 12:15:30.25</td>
<td>111100112153025</td>
<td>1633090530.3</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

**Restriction:** 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
input @1 bda b8601da.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Formatted Using B8601DA Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>20120915</td>
<td>19251</td>
<td>20120915</td>
</tr>
<tr>
<td>2012</td>
<td>18993</td>
<td>20120101</td>
</tr>
</tbody>
</table>

See Also
“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

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

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 B8601DJ informat reads a date and time value that is specified using the Java date and time notation \(yyyymmddhhmms<ffffff>\):

- \(yyyy\) is a four-digit year between 0000 and 9999.
- \(mm\) is a two-digit month (zero padded) between 01 and 12.
- \(dd\) is a two-digit day of the month (zero padded) between 01 and 31.
- \(hh\) is a two-digit hour (zero padded) between 00 and 23.
- \(mm\) is a two-digit minute (zero padded) between 00 and 59.
- \(ss\) is a two-digit second (zero padded) between 00 and 59.
- \(ffffff\) are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

**Comparisons**

The B8601DJ informat reads a date and time value that does not include a T to separate the date from the time.

Java date and time values do not include a T. For example, the date January 1, 2011 at 4:30:25 a.m. is written as 20110101043025.

ISO 8601 date and time values include a T. For example, the date January 1, 2011 at 4:30:25 a.m. is written as 20110101T043025.

**Example**

```plaintext
input @1 bdj b8601dj.;
```
B8601DNw. Informat

Reads date values that are specified using the ISO 8061 basic notation `yyyyymmdd` and returns SAS datetime values where the time portion of the value is 000000.

**Categories:** Date and Time
ISO 8061

**Alignment:** Left

**Alias:** ND8601DNw

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

**Supports:** ISO 8061 Element 5.2.1.1, complete representation

### Syntax

`B8601DNw`.  

### Syntax Description

**w**

specifies the width of the input field.

**Default**

10

**Requirement**

The width of the input field must be 10.

### Details

The B8601DN informat reads date values that are specified using the ISO 8061 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

```plaintext
input @1 bdn b8601dn.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>20120915</td>
<td>1663286400</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

---

**B8601DTw.d Informat**

Reads datetime values that are specified using the ISO 8601 basic notation `yyyyymdddThhmmss<ffffff>`.

**Categories:** Date and Time

**Alignment:** Left

**Aliases:**

- B8601LXw.d
- ND8601DTw.d

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

**Syntax**

```plaintext
B8601DTw.d
```

**Syntax Description**

`w`

- specifies the width of the input field.
  - **Default:** 19
  - **Range:** 19–26

`d`

- specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.
  - **Default:** 0
  - **Range:** 0–6

**Details**

The B8601DT informat reads datetime values that are specified in the ISO 8601 basic datetime notation `yyyyymdddThhmmss<ffffff>`:
yyyy
  is a four-digit year.

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

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

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

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

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

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

If the month or day values are omitted, SAS uses a value of 1 for the month or day. If the
hour, minute, or second values are omitted, SAS uses a value of 0 for the hour, minute,
or second.

Example

input @1 bdt b8601dt.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Formatted Using B8601DT Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>20120915T155300</td>
<td>1663343580</td>
<td>20120915T155300</td>
</tr>
<tr>
<td>20120101T000000</td>
<td>1640995200</td>
<td>20120101T000000</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

B8601DXw. Informat

Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, reads
the local date and time by using the ISO 8601 datetime and time zone basic notation yyyyymmddThhmmss
+|-hhmm

See: The B8601DXw. informat uses the B8601DZw.d informat to read data. See
“B8601DZw.d Informat” on page 289.

Syntax

B8601DXw.
B8601DZw.d Informat

Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation `yyyymmddThhmss[+-]hhmm` or `yyyymmddThhmss<ffffff>Z`.

**Categories:** Date and Time
ISO 8601

**Alignment:** Left

**Aliases:** B8601DXw.d
ND8601DZw.d

**Restriction:** 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 20–35

\(d\)

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

Default 0
Range 0–6

**Details**

UTC values specify a time and a time zone based on the zero meridian in Greenwich, England. The B8601DZ informat reads datetime values that are specified in one of the following ISO 8601 basic datetime notations:

- `yyyymmddThhmss[+-]hhmm`
- `yyyymmddThhmss<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.
is a two-digit hour (zero padded) between 00 and 24.

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

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

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

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

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

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

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

---

**Example**

```plaintext
input @1 bdz b8601dz.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>20120915T155300+0500</td>
<td>1663125580</td>
</tr>
</tbody>
</table>

---

**See Also**

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

---

**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.

**See:** The B8601LXw.d informat uses the B8601DTw.d informat to read data. See “B8601DTw.d Informat” on page 287.

---

**Syntax**

**B8601LXw.d**
B8601TMw.d Informat

Reads time values that are specified using the ISO 8601 basic notation hhmmss<ffffff>.

Categories: Date and Time
ISO 8601

Alignment: Left

Alias: ND8601TMw.d

Restriction: UTC time zone offset values are not supported.

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

Syntax

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 8601 basic time notation hhmmss<ffffff>:

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

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

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

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

Example
input @1 btm b8601tm;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>155300</td>
<td>57180</td>
</tr>
</tbody>
</table>

**See Also**

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

---

**B8601TXw. Informat**

Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, reads the local time by using the ISO 8601 basic time notation \textit{hhmmss}\texttt{+|–}hhmm.

See:  The B8601TXw. informat uses the B8601TZw.d informat to read data. See “B8601TZw.d Informat” on page 292.

**Syntax**

B8601TXw.

---

**B8601TZw.d Informat**

Reads time values that are specified using the ISO 8601 basic time notation \textit{hhmmss}<fff><|–hhmm or \textit{hhmmss}<ffffff>Z.

**Categories:** Date and Time  
ISO 8601  
**Alignment:** Left  
**Aliases:** B8601TXw.d  
ND8601TZw.d  
**Restriction:** UTC time zone offset values are not supported.  
**Supports:** ISO 8601 Element 5.3.1.1, complete representation

**Syntax**

B8601TZw.d

**Syntax Description**

\textit{w}  
specifies the width of the input field.
Default 14
Range 9–20

d
(optional) specifies the number of digits to the right of the decimal point in the seconds value.
Default 0
Range 0–6

Details
UTC time values specify a time and a time zone based on the zero meridian in Greenwich, England. The B8601TZ informat reads time values that are specified in the following ISO 8601 basic time notations:

- \( hhmmss<ffffff>\pm hhmm \)
- \( hhmmss<ffffff>Z \)

\( hh \)
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.

\( fffffff \)
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 \( \pm hhmm \) (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

```
input @1 btz b8601tz.;
```
### BINARYw.d Informat

Converts positive binary values to integers.

**Category:** Numeric

#### Syntax

`BINARY w.d`

#### Syntax Description

- **w**
  - Specifies the width of the input field.
  - **Default:** 8
  - **Range:** 1–64

- **d**
  - Specifies the power of 10 by which to divide the value. SAS uses the `d` value even if the data contain decimal points. This argument is optional.
  - **Range:** 0–31

#### Details

- Use only the character digits 1 and 0 in the input, with no embedded blanks.
- `BINARY w.d` ignores leading and trailing blanks.
- `BINARY w.d` cannot read negative values. It treats all input values as positive (unsigned).

#### Example

```plaintext
input @1 value binary8.1;
```
**BITSw.d Informat**

Extracts bits.

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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.
- Default: 1
- Range: 1–64

**d**

- Specifies the zero-based offset.
- Range: 0–63

### Details

The BITSw.d informat extracts particular bits from an input stream and assigns the numeric equivalent of the extracted bit string to a variable. Together, the **w** and **d** values specify the location of the string 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

```
input @1 value bits4.1;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001111</td>
<td>1.5</td>
</tr>
</tbody>
</table>

---

**Table:**

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001111</td>
<td>1.5</td>
</tr>
</tbody>
</table>
The EBCDIC binary code for a capital B is 11000010, and the ASCII binary code is 01000010.

The input pointer moves to column 2 ($d=1$). Then the INPUT statement reads four bits ($w=4$), which is the bit string 1000, and stores the numeric value 8, which is equivalent to this binary combination.

**BZw.d Informat**

Converts blanks to 0s.

**Category:** Numeric

**Syntax**

\[ BZw.d \]

**Syntax Description**

$w$

specifies the width of the input field.

- **Default:** 1
- **Range:** 1–32

$d$

specifies the power of 10 by which to divide the value. If the data contain decimal points, the $d$ value is ignored. This argument is optional.

- **Range:** 0–31

**Details**

The 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 BZ\(w.d\) informat converts trailing or embedded blanks to 0s. If you do not want to convert trailing blanks to 0s (for example, when reading values in \(E\) notation), use either the \(w.d\) informat or the \(Ew.d\) informat instead.

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<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.

**Category:** Column Binary

**Syntax**

\[CBw.d\]

**Syntax Description**

\(w\)

specifies the width of the input field.

- **Range:** 1–32

\(d\)

specifies the power of 10 by which to divide the value. SAS uses the \(d\) value even if the data contain decimal points. This argument is optional.

**Details**

Column-binary data storage compresses data so that more than 80 items of data can be stored on a single “virtual” punch card.

The \(CBw.d\) informat reads standard numeric values from column-binary files and translates the data into standard binary format.

SAS first stores each column of column-binary data that you read with \(CBw.d\) in two bytes and ignores the two high-order bits of each byte. If the punch codes are valid, then SAS stores the equivalent numeric value in the variable that you specify. If the
combinations are not valid, then SAS assigns the variable a missing value and sets the automatic variable _ERROR_ to 1.

Example: Examples

```sas
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


Informats

- “$CBw. Informat” on page 257
- “PUNCH.d Informat” on page 343
- “ROWw.d Informat” on page 348

COMMAw.d Informat

Removes embedded characters.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Numeric</th>
</tr>
</thead>
</table>

| Alias: | DOLLARw.d |

Syntax

COMMAw.d

Syntax Description

\[ w \]

- specifies the width of the input field.

  Default 1

  Range 1–32

\[ d \]

- specifies the power of 10 by which to divide the value. If the data contain decimal points, the \[ d \] value is ignored. This argument is optional.

  Range 0–31
Details

The COMMA\textit{w.d} informat reads numeric values and removes embedded commas, blanks, dollar signs, percent signs, hyphens, and close parentheses from the input data. The COMMA\textit{w.d} informat converts an open parenthesis at the beginning of a field to a minus sign.

Comparisons

The COMMA\textit{w.d} informat operates like the COMMAX\textit{w.d} informat, but it reverses the roles of the decimal point and the comma. This convention is common in European countries.

Example

\begin{verbatim}
input @1 x comma10.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000000</td>
<td>1000000</td>
</tr>
<tr>
<td>(500)</td>
<td>-500</td>
</tr>
</tbody>
</table>

COMMAX\textit{w.d} Informat

Removes embedded periods, blanks, dollar signs, percent signs, hyphens, and closing parenthesis from the input data. An open parenthesis at the beginning of a field is converted to a minus sign. The COMMAX informat reverses the roles of the decimal point and the comma.

Category: Numeric

Alias: DOLLARX\textit{w.d}

Syntax

\textsc{COMMAX}\textit{w.d}

Syntax Description

\textit{w} specifies the width of the input field.

\begin{verbatim}
Default: 1
Range: 1–32
\end{verbatim}

\textit{d} specifies the power of 10 by which to divide the value. If the data contain a comma, which represents a decimal point, the \textit{d} value is ignored. This argument is optional.
Details
The COMMAXw.d informat reads numeric values and removes embedded periods, blanks, dollar signs, percent signs, hyphens, and close 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 COMMAw.d informat, but it reverses the roles of the decimal point and the comma. This convention is common in European countries.

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

<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 *ddmmyyy* or *ddmmyyyyy*.

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 4-digit year.
Details

The date values must be in the form `ddmmmyy` or `ddmmmyyyy`:

- `dd` is an integer between 01 and 31 that represents the day of the month.
- `mmm` is the first three letters of the month name.
- `yy` or `yyyy` is a two-digit or four-digit integer that represents the year.

You can separate the year, month, and day values by blanks or by special characters. Make sure the width of the input field allows space for blanks and special characters.

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

Example

```
input calendar_date datell.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>16mar12</td>
<td>19068</td>
</tr>
<tr>
<td>16 mar 12</td>
<td>19068</td>
</tr>
<tr>
<td>16-mar-2012</td>
<td>19068</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “DATEw. Format” on page 89

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 `ddmmmyy hh:mm:ss.ss` or `ddmmmyyyy hh:mm:ss.ss`.

Category: Date and Time
Syntax

\textbf{DATETIME}_{w}.

\textbf{Syntax Description}

\(w\)

specifies the width of the input field.

\begin{itemize}
  \item \textbf{Default}: 18
  \item \textbf{Range}: 13–40
\end{itemize}

Details

The datetime values must be in the following form: \textit{ddmmmyy} or \textit{ddmmmyyyy}, followed by a blank or special character, followed by \textit{hh:mm:ss.ss} (the time):

\begin{itemize}
  \item \textit{dd} is an integer between 01 and 31 that represents the day of the month.
  \item \textit{mmm} is the first three letters of the month name.
  \item \textit{yy} or \textit{yyyy} is a two-digit or four-digit integer that represents the year.
  \item \textit{hh} is an integer between 00 and 23 that represents hours.
  \item \textit{mm} is an integer between 00 and 59 that represents minutes.
  \item \textit{ss.ss} is the number of seconds ranging from 00–59 with the fraction of a second following the decimal point.
\end{itemize}

DATETIME\(_{w}\). requires values for both the date and the time. However, the \textit{ss.ss} portion is optional.

\textit{Note:} SAS interprets a two-digit year as belonging to the 100-year span that is defined by the \texttt{YEARCUTOFF=} system option.

\textit{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 \textit{dd-mmm-yy<yy>} \textit{hh:mm:ss.ss} AM/PM, and the date and time can be separated by a special character.

The MDYAMP\(_{w.d}\) in format reads datetime values with optional separators in the form \textit{mm-dd-yy<yy>} \textit{hh:mm:ss.ss} AM | PM, and requires a space between the date and the time.

The YMDDTT\(_{w.d}\) informat reads datetime values with required separators in the form \textit{<yy>}\textit{yy-mm-dd}/\textit{hh:mm:ss.ss}.

Example

\begin{verbatim}
  input date_and_time datetime20.;
\end{verbatim}
### Data Line

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>16mar12:11:23:07.4</td>
<td>1647516187.4</td>
</tr>
<tr>
<td>16mar2012/11:23:07.4</td>
<td>1647516187.4</td>
</tr>
<tr>
<td>16mar2012/11:23 PM</td>
<td>1647559380.0</td>
</tr>
</tbody>
</table>

### See Also

- “About SAS Date, Time, and Datetime Values” in *SAS Language Reference: Concepts*

### Formats:

- “DATEw. Format” on page 89
- “DATETIMEw.d Format” on page 92
- “TIMEw.d Format” on page 184

### Functions:

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

### Informats:

- “DATEw. Informat” on page 300
- “MDYAMPMw.d Informat” on page 325
- “TIMEw. Informat” on page 369
- “YMDDTTMw.d Informat” on page 383

### System Options:

- “YEARCUTOFF= System Option” in *SAS System Options: Reference*

---

**DDMMYYw. Informat**

Reads date values in the form *ddmmyy<yy>* or *dd-mm-yy<yy>*; where a special character, such as a hyphen (-), period (.), or slash (/), separates the day, month, and year; the year can be either 2 or 4 digits.  

**Category:** Date and Time

**Syntax**

```
DDMMYYw.
```
**Syntax Description**

`w` specifies the width of the input field.

- **Default**: 6
- **Range**: 6–32

**Details**

The date values must be in the form `ddmmyy<yy>` or `ddxmmxyy<yy>`:

- `dd` is an integer between 01 and 31 that represents the day of the month.
- `mm` is an integer between 01 and 12 that represents the month.
- `yy` or `yyyy` is a two-digit or four-digit integer that represents the year.
- `x` is a separators that can be any special character or a blank.

If you use separators, place them between all the values. Blanks can also be placed before and after the date. Make sure the width of the input field allows space for blanks and special characters.

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

**Example**

```plaintext
input calendar_date ddmmyy10.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>160308</td>
<td>19068</td>
</tr>
<tr>
<td>16/03/08</td>
<td>19068</td>
</tr>
<tr>
<td>16-03-2008</td>
<td>19068</td>
</tr>
<tr>
<td>16 03 2008</td>
<td>19068</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “DATEw. Format” on page 89
- “DDMMYYw. Format” on page 95
- “MMDDYYw. Format” on page 137
• “YYMMDw. Format” on page 209

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

Informats:
• “DATEw. Informat” on page 300
• “MMDDYYw. Informat” on page 327
• “YYMMDw. Informat” on page 385

System Options:
• “YEARCUTOFF= System Option” in *SAS System Options: Reference*

---

**E8601DAw. Informat**

Reads date values that are specified using the ISO 8601 extended notation *yyyy-mm-dd*.

**Categories:** Date and Time

ISO 8601

**Alignment:** Left

**Alias:** 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.

<table>
<thead>
<tr>
<th>Default</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>The width of the input field must be 10.</td>
</tr>
</tbody>
</table>

**Details**

The E8601DA informat reads date values that are specified in the ISO 8601 extended date notation *yyyy-mm-dd*:

- *yyyy* is a four-digit year.
- *mm* is a two-digit month (zero padded) between 01 and 12.
$dd$

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

**Example**

```
input eda e8601da.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-09-15</td>
<td>19251</td>
</tr>
</tbody>
</table>

**See Also**

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

---

**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:** 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**

`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 $yyyy-mm-dd$ and returns the date in a SAS datetime value:

$yyyy$

is a four-digit year.
**E8601DTw.d Informat**

Reads datetime values that are specified using the ISO 8601 extended notation `yyyy-mm-ddThh:mm:ss.<fffffff>`.

- **Categories:** Date and Time
 ipe ISO 8601
- **Alignment:** Left
- **Aliases:** E8601LXw.d
  IS8601DTw.d
- **Restriction:** UTC time zone offset values are not supported.
- **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:** 19–26

**d**

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

This argument is optional.
The E8601DT informat reads datetime values that are specified using the ISO 8601 extended datetime notation \texttt{yyyy-mm-ddThh:mm:ss.<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.
- \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**

```
input @1 edt e8601dt.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{2012-09-15T15:53:00}</td>
<td>\texttt{1663343580}</td>
</tr>
</tbody>
</table>

**See Also**

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

---

**E8601DXw. Informat**

Adjusts a Coordinated Universal Time (UTC) datetime value to the user local date and time. Then, reads the local date and time by using the ISO 8601 datetime and time zone extended notation \texttt{yyyy-mm-ddThh:mm:ss+[-]hh:mm}.

**See:** The E8601DXw. informat uses the E8601DZw.d informat to read data. See “E8601DZw.d Informat” on page 309.
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.<ffffff>` or `yyyy-mm-ddThh:mm:ss.<ffffff>Z`.  

**Categories:** 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 the following ISO 8601 extended datetime notations:

- `yyyy-mm-ddThh:mm:ss.<ffffff>+|–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.

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

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

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

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

Z
  indicates that the time is UTC time at the zero meridian (Greenwich, England).

Example

<table>
<thead>
<tr>
<th>Input Statement</th>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----1----+----2----+----3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>input edz e8601dz.;</th>
<th>2012-09-15T15:53:00Z</th>
<th>1663343580</th>
</tr>
</thead>
<tbody>
<tr>
<td>input edz e8601dz28.2;</td>
<td>2012-09-15T15:53:00+03:00</td>
<td>1663332780</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237

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.

See: The E8601LXw. informat uses the E8601DTw.d informat to read data. See “E8601DTw.d Informat” on page 307.

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

**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 data follows on the same line of data.

d

specifies the number of digits to the right of the decimal point in the value for the lowest-order component. This argument is optional.

- **Default:** 0
- **Range:** 0–6

**Details**

UTC values specify a time and a time zone based on the zero meridian in Greenwich, England. The E8601LZ informat reads UTC time values that are specified in one of the following ISO 8601 extended time notations and returns a SAS time value for the local time:

- `hh:mm:ss.<ffffff>+|−hh:mm`
- `hh:mm:ss.<ffffff>Z`

`hh`

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

`mm`

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

`ss`

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

```plaintext
input elz e8601lz.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>09:13:21+02:00</td>
<td>26001</td>
</tr>
<tr>
<td>23:43:44Z</td>
<td>85424</td>
</tr>
</tbody>
</table>

**See Also**

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on
page 237

---

**E8601TMw.d Informat**

Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.<ffffff>.

**Categories:** Date and Time

**Alignment:** Left

**Alias:** IS8601TMw.d

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

**Supports:** ISO 8601 Elements 5.3.1.1 and 5.3.1.3, complete representation and representation
of decimal fractions
Syntax

\texttt{E8601TMw.d}

\textbf{Syntax Description}

\textit{w}

specifies the width of the input field.

Default: 8

Range: 8–15

\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

\textbf{Details}

The E8601TM informat reads time values that are specified using the ISO 8601 extended time notation \texttt{\textasciitilde{\text{hh:mm:ss.<ffffff>}}}:  

\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.

\textit{ffffff}

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

\textbf{Example}

\begin{verbatim}
input @1 etm e8601tm.;
\end{verbatim}

<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>

\textbf{See Also}

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 237
**E8601TXw. Informat**

Adjusts a Coordinated Universal Time (UTC) value to the user local time. Then, reads the local time by using the ISO 8601 extended time notation `hh:mm:ss+-hh:mm`.

See: The E8601TXw. informat uses the E8601TZw.d informat to read data. See "E8601DZw.d Informat" on page 309.

**Syntax**

E8601TXw.

---

**E8601TZw.d Informat**

Reads time values that are specified using the ISO 8601 extended time notation `hh:mm:ss+-hh:mm.<ffffff>` or `hh:mm:ssZ`.

**Categories:** Date and Time

**Alignment:** Left

**Aliases:** E8601TXw.d

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

**Syntax**

E8601TZw.d

**Syntax Description**

-w

specifies the width of the input field.

<table>
<thead>
<tr>
<th>Default</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>9–20</td>
</tr>
<tr>
<td>Requirement</td>
<td>To read a time with the Z time zone indicator, the width of the input field must be 9 if data follows on the same line of data.</td>
</tr>
</tbody>
</table>

-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.

<table>
<thead>
<tr>
<th>Default</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0–6</td>
</tr>
</tbody>
</table>
UTC time values specify a time and a time zone based on the zero meridian in Greenwich, England. The E8601TZ informat reads UTC time values that are specified in one of the following ISO 8601 extended notations:

- $hh:mm:ss\pm|–hh:mm.<fffff>$
- $hh:mm:ssZ$

$hh$

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.

$fffff$

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. Note that the offset must be $\pm|–hh:mm$ (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 $\pm|–hh$ 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**

```sas
input @1 etz e8601tz.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------+-----+-----+</td>
<td></td>
</tr>
<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 860 Basic and Extended Notations” on page 237
FLOATw.d Informat
Reads a native single-precision, floating-point value and divides it by 10 raised to the dth power.

Category: Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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 four-byte floating-point number is the same as a truncated eight-byte floating-point number. However, in operating environments that use the IEEE floating-point standard, such as the IBM PC-based operating environments and most UNIX platforms, a four-byte floating-point number is not the same as a truncated double. Therefore, the RB4. informat does not produce the same results as FLOAT4. Floating-point representations other than IEEE might have this same characteristic. Values read with FLOAT4 typically come from some other external program that is running in your operating environment.

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

<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>
</tbody>
</table>
**Language** | **Float Notation**
---|---
PL/I | FLOAT BIN(21)

**Example**

```plaintext
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.

**Category:** Numeric

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

**Syntax**

```plaintext
HEXw.
```

**Syntax Description**

`w`

Specifies the field width of the input value and also specifies whether the final value is fixed-point or floating-point.

**Default** 8

**Range** 1–16

**Tip** If `w<16`, HEXw. converts the input value to positive integer binary values, treating all input values as positive (unsigned). If `w` is 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

input @1 x hex3. @5 y hex16.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>88F 4152000000000000</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 \texttt{hh:mm:ss} or \texttt{hhmmss}.

\textbf{Category:} Date and Time

\textbf{Syntax}

\texttt{HHMMSSw.}

\textbf{Syntax Description}

\texttt{w}

specifies the width of the input field.

\textbf{Default} \hspace{1cm} 8

\textbf{Range} \hspace{1cm} 1–20

\textbf{Details}

The HHMMSSw. informat reads SAS time values in one of the following forms:

- \texttt{hh:mm:ss}
- \texttt{hhmmss}

\texttt{hh}

is an integer that represents the number of hours.

\texttt{:}

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.

### Example

```r
input tm hhmmss;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Formatted with TIMEw.</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>82800</td>
<td>23:00:00</td>
</tr>
<tr>
<td>12:45:44</td>
<td>45344</td>
<td>12:45:44</td>
</tr>
<tr>
<td>2358444</td>
<td>851084</td>
<td>236:24:44</td>
</tr>
<tr>
<td>17:35</td>
<td>63300</td>
<td>17:35:00</td>
</tr>
<tr>
<td>12:3400</td>
<td>247200</td>
<td>68:40:00</td>
</tr>
</tbody>
</table>

### See Also

Informats:
**IBw.d Informat**

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

- **Category:** Numeric
- **Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

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

**Syntax**

`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, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 231.

**Comparisons**

The IBw.d and PIBw.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 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 or Little Endian Platforms” on page 232.
To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 9.

Example

You can use the INPUT statement and specify the IB informat. However, these examples use the informat with the INPUT function, where binary input values are described using a hexadecimal literal.

```sas
x=input('0080'x,ib2.);
y=input('8000'x,ib2.);
```

<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>128</td>
<td>-32768</td>
</tr>
<tr>
<td>put y=;</td>
<td>-32768</td>
<td>128</td>
</tr>
</tbody>
</table>

See Also

Informats:

- “IBRw.d Informat” on page 321

---

**IBRw.d Informat**

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

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**Syntax**

`IBRw.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 IBRw.d informat reads integer binary (fixed-point) values, including negative values that are represented in two's complement notation. IBRw.d reads integer binary values that are generated by and for Intel and DEC platforms. Use IBRw.d to read integer binary data from Intel or DEC environments in other operating environments. The IBRw.d informat in SAS code allows for a portable implementation for reading the data in any operating environment.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 231.

Comparisons
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.

On Intel and DEC operating environments, the IBw.d and IBRw.d informats are equivalent.

To view a table that shows the type of informat to use with big endian and little endian integers, see “Reading Data Generated on Big Endian or Little Endian Platforms” on page 232.

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

Example
You can use the INPUT statement and specify the IBR informat. However, in these examples that we use the informat with the INPUT function, where binary input values are described using a hexadecimal literal.

```sas
x=input('0100'x,ibr2.);
y=input('0001'x,ibr2.);
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result on BigEndian Platforms</th>
<th>Result on LittleEndian 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>

See Also

Informats:
- “IBw.d Informat” on page 320

IEEEw.d Informat
Reads an IEEE floating-point value and divides it by 10 raised to the d th power.
Category: Numeric
Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**Syntax**

IEEE\(w.d\)

**Syntax Description**

\(w\)

specifies the width of the input field.

- **Default**: 8
- **Range**: 2–8

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

\(d\)

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

**Details**

The IEEE\(w.d\) informat is useful in operating environments where IEEE is the floating-point representation that is used. In addition, you can use the IEEE\(w.d\) informat to read files that are created by programs on operating environments that use the IEEE floating-point representation.

Typically, programs generate IEEE values in single precision (4 bytes) or double precision (8 bytes). Truncation is performed by programs solely to save space on output files. Machine instructions require that the floating-point number be of one of the two lengths. The IEEE\(w.d\) informat allows other lengths, which enables you to read data from files that contain space-saving truncated data.

**Example**

```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>3FP000000</td>
<td>1</td>
</tr>
</tbody>
</table>

* The data lines are hexadecimal representations of binary numbers that are stored in IEEE format.
The first INPUT statement reads the first data line, and the second INPUT statement reads the next data line.

**JULIANw. Informat**

Reads Julian dates in the form *yyddd* or *yyyyddd*.

**Category:** Date and Time

**Syntax**

```JULIANw.```

**Syntax Description**

`w`

specifies the width of the input field.

- **Default:** 5
- **Range:** 5–32

**Details**

The date values must be in one of the following forms:

- *yyddd*
- *yyyyddd*

*yy* or *yyyy*

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

*dd* or *ddd*

is an integer from 01–365 that represents the day of the year.

Julian dates consist of strings of contiguous numbers, which means that zeros must pad any space between the year and the day values.

Julian dates that contain year values before 1582 are invalid for the conversion to Gregorian dates.

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

**Example**

```input julian_date julian7.;```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12076</td>
<td>19068</td>
</tr>
</tbody>
</table>
The input values correspond to the 76th day of 2012, which is March 16.

See Also

Formats:
- “JULIANw. Format” on page 134

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

MDYAMPMw.d Informat

Reads datetime values in the form mm-dd-yy<yy> hh:mm:ss.ss AM|PM, where a special character such as a hyphen (-), period (.), slash (/), or colon (:) separates the month, day, and year; the year can be either 2 or 4 digits.

Category: Date and Time
Alignment: Right
Requirement: A space must separate the date and the time.
Note: The default time period is AM.

Syntax

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
Details
The MDYAMPMw.d format reads SAS datetime values in the form \texttt{mm-dd-yy<yy> hh:mm<ss<ss>> AM | PM}: \\
\texttt{mm} \text{" is an integer between 01 and 12 that represents the month.\text{"} } \\
\texttt{dd} \text{" is an integer between 01 and 31 that represents the day of the month.\text{"} } \\
\texttt{yy or yyyy} \text{" specifies a two-digit or four-digit integer that represents the year.\text{"} } \\
\texttt{hh} \text{" is an integer between 00 and 23 that represents hours.\text{"} } \\
\texttt{mm} \text{" is an integer between 00 and 59 that represents minutes.\text{"} } \\
\texttt{ss.ss} \text{" is the number of seconds that range from 00–59 with the fraction of a second following the decimal point.\text{"} } \\
\textbf{Requirement:} If a fraction of a second is specified, the decimal point can be represented only by a period and is required.
\texttt{AM | PM} \text{" specifies either the time period 00:01–12:00 noon (AM) or the time period 12:01–12:00 midnight ( PM) } \\
\text{" 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.\text{"} \\

Comparisons
The MDYAMPMw.d informat reads datetime values with optional separators in the form \texttt{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 \texttt{dd-mm-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 \texttt{<yy>yy-mm-dd/hh:mm:ss.ss}.

Example
\texttt{input @1 dt mdyampm25.2.;}

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>09.15.2012 03:53:00 pm</td>
<td>1663343580</td>
</tr>
</tbody>
</table>
See Also

Informats:

- “DATETIMEw. Informat” on page 301
- “YMDDTTMw.d Informat” on page 383

**MMDDYYw. Informat**

Reads date values in the form *mmddyy* or *mmddyyyy*.

**Category:** Date and Time

**Syntax**

`MMDDYYw.`

**Syntax Description**

`w` specifies the width of the input field.

Default: 6

Range: 6–32

**Details**

The date values must be in one of the following forms:

- *mmddyy*
- *mmddyyyy*

*mm* is an integer between 01 and 12 that represents the month.

*dd* is an integer between 01 and 31 that represents the day of the month.

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

You can separate the month, day, and year fields by blanks or by special characters. However, if you use delimiters, place them between all fields in the value. Blanks can also be placed before and after the date.

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

    input calendar_date mmdyy8.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>031612</td>
<td>19068</td>
</tr>
<tr>
<td>03/16/12</td>
<td>19068</td>
</tr>
<tr>
<td>03 16 12</td>
<td>19068</td>
</tr>
<tr>
<td>03162012</td>
<td>19068</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DATEw. Format” on page 89
- “DDMMYYw. Format” on page 95
- “MMDDYYw. Format” on page 137
- “YYMMDDw. Format” on page 209

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 300
- “DDMMYYw. Informat” on page 303
- “YYMMDDw. Informat” on page 385

System Options:

- “YEARCUTOFF= System Option” in SAS System Options: Reference

MONYYw. Informat

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

Category: Date and Time
Syntax

MONYYw.

**Syntax Description**

w

specifies the width of the input field.

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

**Details**

The date values must be in one of the following forms:

- \( mmmyy \)
- \( mmmyyyy \)

\( mmm \)

is the first three letters of the month name.

\( yy \) or \( yyyy \)

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

A value read with the MONYYw. informat results in a SAS date value that corresponds to the first day of the specified month.

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

**Example**

```r
input month_and_year monyy7.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>mar 12</td>
<td>19053</td>
</tr>
<tr>
<td>mar2012</td>
<td>19053</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “DDMMYYw. Format” on page 95
- “MMDDYYw. Format” on page 137
- “MONYYw. Format” on page 147
- “YYMMDDw. Format” on page 209

**Functions:**
MSECw. Informat
Reads TIME MIC values.

Category: Date and Time

Syntax
MSECw.

Syntax Description
w

specifies the width of the input field.

Requirement w must be 8 because the OS TIME macro or the STCK System/370 instruction on IBM mainframes each return an eight-byte value.

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

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

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

Example

input btime msec8.;
The data line is a hexadecimal representation of a binary 8-byte time-of-day clock value. Each byte occupies one column of the input field. The result is a SAS time value corresponding to 5:26:58.41 p.m.

See Also

Informats:
- “TODSTAMPw. Informat” on page 371

NUMXw.d Informat
Reads numeric values with a comma in place of the decimal point.

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 contain 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

input @1 x numx1.;
### OCTALw.d Informat

Converts positive octal values to integers.

**Category:** Numeric

### Syntax

OCTAL\(w.d\)

### Syntax Description

**\(w\)**

specifies the width of the input field.

- **Default:** \(3\)
- **Range:** \(1–24\)

**\(d\)**

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

- **Range:** \(1–31\)
- **Restriction:** must be greater than or equal to the \(w\) value.

### Details

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

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

### Example

<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:
- “NUMXw.d Format” on page 149
- “w.d Format” on page 192
input @1 value octal3.1;

<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 are stored in IBM packed decimal format.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction:</td>
<td>List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.</td>
</tr>
</tbody>
</table>
| See:        | “PDw.d Informat: UNIX” in SAS Companion for UNIX Environments  
“PDw.d Informat: Windows” in SAS Companion for Windows  
“PDw.d Informat: z/OS” in SAS Companion for z/OS |

**Syntax**

`PDw.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 PDw.d informat is useful because many programs write data in packed decimal format for storage efficiency, fitting two digits into each byte and using only a half byte for a sign.

*Note:* Different operating environments store packed decimal values in different ways. However, PDw.d reads packed decimal values with consistent results if the values are created on the same type of operating environment that you use to run SAS.

The PDw.d format writes missing numerical data as -0. When the PDw.d informat reads -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>

Examples

**Example 1: Reading Packed Decimal Data**

```plaintext
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**

```plaintext
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 on an IBM mainframe operating environment. Each byte occupies one column of the input field. The result is a SAS date value that corresponds to December 25, 2010.

**PDJULGw. Informat**

Reads packed Julian date values in the hexadecimal form yyyydddF for IBM.

**Category:** Date and Time
Syntax

PDJULGw.

Syntax Description

w

specifies the width of the input field.

Default 4
Range 4

Details

The PDJULGw. informat reads IBM packed Julian date values in the form of yyyydddF:

yyyy

is the two-byte representation of the four-digit Gregorian year.

ddd

is the one-and-a-half byte representation of the three-digit integer that corresponds to the Julian day of the year, 1–365 (or 1–366 for leap years).

F

is the half byte that contains all binary 1s, which assigns the value as positive.

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

Example

input date pdjulg4.;

<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 133
- “JULIANw. Format” on page 134
- “PDJULGw. Format” on page 153
- “PDJULIw. Format” on page 154

Functions:

- “DATEJUL Function” in SAS Functions and CALL Routines: Reference
- “JULDATE Function” in SAS Functions and CALL Routines: Reference
Informats:
• “JULIANw. Informat” on page 324
• “PDJULIw. Informat” on page 336

System Options:
• “YEARCUTOFF= System Option” in SAS System Options: Reference

PDJULIw. Informat
Reads packed Julian dates in the hexadecimal format ccyydddF for IBM.

Syntax
PDJULIw.

Syntax Description
w
specifies the width of the input field.
Default 4
Range 4

Details
The PDJULIw. informat reads IBM packed Julian date values in the form ccyydddF:
cc
is the one-byte representation of a two-digit integer that represents the century.

yy
is the one-byte representation of a two-digit integer that represents the year. The PDJULIw informat makes an adjustment to the one-byte century representation by adding 1900 to the two-byte ccyy value in order to produce the correct four-digit Gregorian year. This adjustment causes ccyy values of 0098 to become 1998, 0101 to become 2001, and 0218 to become 2118.

ddd
is the one-and-a-half bytes representation of the three-digit integer that corresponds to the Julian day of the year, 1–365 (or 1–366 for leap years).

F
is the half byte that contains all binary 1s, which assigns the value as positive.

Example
input date pdjuli4.;
**Data Line** | **Result** |
--- | --- |
--------- | ---- |
0099001F | 14245 |
0112015F | 19007 |

* SAS date value 14245 is January 1, 1999. SAS date value 19007 is January 15, 2012.

**See Also**

**Formats:**
- “JULDAYw. Format” on page 133
- “JULIANw. Format” on page 134
- “PDJULGw. Format” on page 153
- “PDJULIw. Format” on page 154

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

**Informats:**
- “JULIANw. Informat” on page 324
- “PDJULGw. Informat” on page 334

**System Options:**
- “YEARCUTOFF= System Option” in *SAS System Options: Reference*

---

**PDTIMEw. Informat**

Reads packed decimal time of SMF and RMF records.

**Category:** Date and Time

**Syntax**

\[
\text{PDTIME}_w.
\]

**Syntax Description**

\[
w
\]

specifies the width of the input field.

**Requirement**  
\( w \) must be 4 because packed decimal time values in RMF and SMF records contain four bytes of information.
Details

The PDTIMEw. informat reads packed decimal time values that are contained in SMF and RMF records that are produced by IBM mainframe systems and converts the values to SAS time values.

The general form of a packed decimal time value in hexadecimal notation is 0hhmmssF:

- 0 is a half byte that contains all 0s.
- hh is one byte that represents two digits that correspond to hours.
- mm is one byte that represents two digits that correspond to minutes.
- ss is one byte that represents two digits that correspond to seconds.
- F is a half byte that contains all 1s.

If a field contains all 0s, PDTIMEw. treats it as a missing value.

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

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.

**Category:** Numeric

**Syntax**

`PERCENTw.d`

**Syntax Description**

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

  Default: 6
Range  1–32

d
specifies the power of 10 by which to divide the value. If the data contain decimal points, the d value is ignored. This argument is optional.
Range  0–31

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

Example

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

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

PIBw.d Informat

Reads positive integer binary (fixed-point) values.

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**See:** “PIBw.d Informat: UNIX” in SAS Companion for UNIX Environments
“PIBw.d Informat: Windows” in SAS Companion for Windows

**Syntax**

PIBw.d

**Syntax Description**

w
specifies the width of the input field.

Default  1
Range  1–8

d
specifies the power of 10 by which to divide the value. This argument is optional.
Range 0–10

Details

All values are treated as positive. PIB\(_w.d\) reads positive integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

*Note:* Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 231.

Comparisons

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIB\(_w.d\) informat treats all values as positive and includes the sign bit as part of the value.
- The PIB\(_w.d\) informat with a width of 1 results in a value that corresponds to the binary equivalent of the contents of a byte. The binary equivalent of the contents of a byte is useful if your data contain values between hexadecimal 80 and hexadecimal FF, where the high-order bit can be misinterpreted as a negative sign.
- The IB\(_w.d\) and PIB\(_w.d\) informats are used to read native format integers. (Native format enables you to read and write values that are created in the same operating environment.) The IBR\(_w.d\) and PIBR\(_w.d\) informats are used to read little endian integers in any operating environment.

To view a table that shows the type of informat to use with big endian and little endian integers, see “Reading Data Generated on Big Endian or Little Endian Platforms” on page 232.

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

Example

You can use the INPUT statement and specify the PIB informat. However, in these examples, we use the informat with the INPUT function, where binary input values are described by using a hexadecimal literal.

\[
\begin{align*}
\text{x=}&\text{input('0100'x,pib2.);} \\
\text{y=}&\text{input('0001'x,pib2.);}
\end{align*}
\]

<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>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

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

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

---

**Syntax**

PIBR\textit{w.d}

**Syntax Description**

\textit{w} specifies the width of the input field.

- **Default:** 1
- **Range:** 1–8

\textit{d} specifies the power of 10 by which to divide the value. This argument is optional.

- **Range:** 0–10

**Details**

All values are treated as positive. PIBR\textit{w.d} reads positive integer binary values that have been generated by and for Intel and DEC operating environments. Use PIBR\textit{w.d} to read positive integer binary data from Intel or DEC environments on other operating environments. The PIBR\textit{w.d} informat in SAS code allows for a portable implementation for reading the data in any operating environment.

**Note:** Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 231.

**Comparisons**

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIBR\textit{w.d} informat treats all values as positive and includes the sign bit as part of the value.

- The PIBR\textit{w.d} informat with a width of 1 results in a value that corresponds to the binary equivalent of the contents of a byte. This is useful if your data contain values between hexadecimal 80 and hexadecimal FF, where the high-order bit can be misinterpreted as a negative sign.

- On Intel and DEC platforms, the PIB\textit{w.d} and PIBR\textit{w.d} informats are equivalent.
The IB\textit{w}.\textit{d} and PIB\textit{w}.\textit{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\textit{w}.\textit{d} and PIBR\textit{w}.\textit{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 or Little Endian Platforms” on page 232.

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

Example

You can use the INPUT statement and specify the PIBR informat. However, these examples use the informat with the INPUT function, where binary input values are described using a hexadecimal literal.

\begin{verbatim}
x=input('0100'x,pibr2.);
y=input('0001'x,pibr2.);
\end{verbatim}

<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>

See Also

Informat

- “PIB\textit{w}.\textit{d} Informat” on page 339

PK\textit{w}.\textit{d} Informat

Reads unsigned packed decimal data.

- **Category:** Numeric
- **Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**Syntax**

PK\textit{w}.\textit{d}

**Syntax Description**

\textit{w}

 specifies the number of bytes of unsigned packed decimal data, each of which contains two digits.
Default  1
Range    1–16

\( d \)
specifies the power of 10 by which to divide the value. This argument is optional.
Range    0–10

Details
Each byte of unsigned packed decimal data contains two digits.

Comparisons
The PK\(_w.d\) informat is the same as the PD\(_w.d\) informat, except that PK\(_w.d\) treats the sign half of the field's last byte as part of the value, not as the sign of the value.

Example

```
input @1 x pk3.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>001234</td>
<td>1234</td>
</tr>
</tbody>
</table>

* The data line is a hexadecimal representation of a binary number stored in unsigned packed decimal form. Each byte occupies one column of the input field.

---

**PUNCH\(d\) Informat**
Reads whether a row of column-binary data is punched.

**Category:** Column Binary

**Syntax**
PUNCH\(_d\)

**Syntax Description**

\( d \)
specifies which row in a card column to read.
Range    1–12

**Details**
Column-binary data storage compresses data so that more than 80 items of data can be stored on a single “virtual” punch card.
This informat assigns the value 1 to the variable if row \(d\) of the current card column is punched, or 0 if row \(d\) of the current card column is not punched. After PUNCH.d reads a field, the pointer does not advance to the next column.

### Example

<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>
<tr>
<td></td>
<td>input x punch.11</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>input x punch0.7</td>
<td>1</td>
</tr>
</tbody>
</table>

* The data line is “virtual” punched card code. The punch card column for the example data has row 12, row 7, and row 8 punched.

### See Also

- “How to Read Column-Binary Data” in *SAS Language Reference: Concepts*

### RBw.d Informat

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

- **Category:** Numeric
- **Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**See:**

- “RBw.d Informat: UNIX” in *SAS Companion for UNIX Environments*
- “RBw.d Informat: Windows” in *SAS Companion for Windows*
- “RBw.d Informat: z/OS” in *SAS Companion for z/OS*

### Syntax

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

### 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:

<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

    input @1 x rb8.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<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 322

---

**RMF\text{DUR}w. Informat**

Reads duration intervals of RMF records.

**Category:** Date and Time

**Syntax**

\texttt{RMF}\text{DUR}w.

**Syntax Description**

\( w \)

specifies the width of the input field.

**Requirement** \( w \) must be 4 because packed decimal duration values in RMF records contain four bytes of information.

**Details**

The \texttt{RMF}\text{DUR}w. informat reads the duration of RMF measurement intervals of RMF records that are produced as packed decimal data by IBM mainframe systems and converts them to SAS time values.

The general form of the duration interval data in an RMF record in hexadecimal notation is \texttt{mmsstttF}:

\[ m \]

is the one-byte representation of two digits that correspond to minutes.

\[ ss \]

is the one-byte representation of two digits that correspond to seconds.

\[ ttt \]

is the one-and-a-half-bytes representation of three digits that correspond to thousandths of a second.

\[ F \]

is a half byte that contains all binary 1s, which assigns the value as positive.

If the field does not contain packed decimal data, then \texttt{RMF}\text{DUR}w. results in a missing value.

**Comparisons**

- Both the \texttt{RMF}\text{DUR}w. informat and the \texttt{RMFSTAMP}w. informat read packed decimal information from RMF records that are produced by IBM mainframe systems.
- The \texttt{RMF}\text{DUR}w. informat reads duration data and results in a time value.
• The RMFSTAMPw. informat reads time-of-day data and results in a datetime value.

Example

```sas
input dura rmfdur4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<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 corresponding to 00:35:52.226.

See Also

Informats:
• “RMFSTAMPw. Informat” on page 347
• “SMFSTAMPw. Informat” on page 367

RMFSTAMPw. Informat

Reads time and date fields of RMF records.

Category: Date and Time

Syntax

```
RMFSTAMPw.
```

Syntax Description

\( w \)

specifies the width of the input field.

Requirement \( w \) must be 8 because packed decimal time and date values in RMF records contain eight bytes of information: four bytes of time data that are followed by four bytes of date data.

Details

The RMFSTAMPw. informat reads packed decimal time and date values of RMF records that are produced by IBM mainframe systems, and converts the time and date values to SAS datetime values.

The general form of the time and date information in an RMF record in hexadecimal notation is 0hhmmssFccydddF:

0

is the half byte that contains all binary 0s.
is the one-byte representation of two digits that correspond to the hour of the day.

mm
is the one-byte representation of two digits that correspond to minutes.

ss
is 1 byte that represents two digits that correspond to seconds.

c
is the one-byte representation of two digits that correspond to the century.

yy
is the one-byte representation of two digits that correspond to the year.

ddd
is the one-and-a-half bytes that contain three digits that correspond to the day of the year.

F
is the half byte that contains all binary 1s.

The century indicators 00 correspond to 1900, 01 to 2000, and 02 to 2100.

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

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

Example
input begin: $hex16.;
y=input(begin, rmfstamp8.);

<table>
<thead>
<tr>
<th>Data Line*</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0142225F2612200F</td>
<td>80550512545</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, 2012, 2:22.25 PM.

ROWw.d Informat
Reads a column-binary field down a card column.

Category: Column Binary
Syntax

ROW_{w.d}

**Syntax Description**

\( w \)

specifies the row where the field begins.

Range \( 0–12 \)

\( d \)

specifies the length in rows of the field.

Default \( 1 \)

Range \( 1–25 \)

**Details**

Column-binary data storage compresses data so that more than 80 items of data can be stored on a single “virtual” punch card.

The ROW_{w.d} informat assigns the relative position of the punch in the field to a numeric variable.

If the field that you specify has more than one punch, then ROW_{w.d} assigns the variable a missing value and sets the automatic variable _ERROR_ to 1. If the field has no punches, then ROW_{w.d} assigns the variable a missing value.

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

**Example**

```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>00</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 257
- “CBw.d Informat” on page 297
- “PUNCH.d Informat” on page 343

---

**S370FFw.d Informat**

Reads EBCDIC numeric data.

**Category:** 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 are 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 one byte per digit. Use S370FFw.d on 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 S370FF\textit{w.d} informat performs the same role for numeric data that the \$EBCDIC\textit{w.d} informat does for character data. That is, on an IBM mainframe system, S370FF\textit{w.d} has the same effect as the standard \textit{w.d} informat. On all other systems, using S370FF\textit{w.d} is equivalent to using \$EBCDIC\textit{w.d} as well as using the standard \textit{w.d} informat.

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>F1F2F3</td>
<td>123</td>
</tr>
<tr>
<td>F2F4F0</td>
<td>240</td>
</tr>
</tbody>
</table>

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

\textbf{S370FIB\textit{w.d} Informat}

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

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**Syntax**

\texttt{S370FIB\textit{w.d}}

**Syntax Description**

\textit{w}

specifies the width of the input field.

 Default 4

 Range 1–8

\textit{d}

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

 Range 0–10
Details

The S370FIBw.d informat reads integer binary (fixed-point) values that are stored in IBM mainframe format, including negative values that are represented in two's complement notation. S370FIBw.d reads integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Use S370FIBw.d for integer binary data that are created in IBM mainframe format for reading in other operating environments.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 231.

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 or Little Endian Platforms” on page 232.

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

Example

You can use the INPUT statement and specify the S370FIB informat. However, this example uses the informat with the INPUT function, where the binary input value is described by using a hexadecimal literal.

```
x=input('0080'x,s370fib2.);
```

<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 352
- “S370FPIBw.d Informat” on page 356

S370FIBUw.d Informat

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

Category: Numeric
Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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 contain decimal points. This argument is optional.
   Range    0–10

Details
The S370FIBUw.d informat reads unsigned integer binary (fixed-point) values that are stored in IBM mainframe format, including negative values that are represented in two’s complement notation. Unsigned integer binary values are the same as integer binary values, except that all values are treated as positive. S370FIBUw.d reads integer binary values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Use S370FIBUw.d for unsigned integer binary data that are created in IBM mainframe format for reading in other operating environments.

Note: Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 231.

Comparisons
• The S370FIBUw.d informat is equivalent to the COBOL notation PIC 9(n) BINARY, where n is the number of digits.
• The S370FIBUw.d and S370FPIBw.d informats 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 or Little Endian Platforms” on page 232.

To view a table that compares integer binary notation in several programming languages, see “Integer Binary Notation and Different Programming Languages” on page 9.
Example
You can use the INPUT statement and specify the S370FIBU informat. However, these examples use the informat with the INPUT function, where binary input values are described by using a hexadecimal literal.

```sas
x=input('7F'x,s370fibu1.);
y=input('F6'x,s370fibu1.);
```

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

See Also

**Informats:**
- “S370FIBw.d Informat” on page 351
- “S370FPIBw.d Informat” on page 356

### S370FPDw.d Informat
Reads packed data in IBM mainframe format.

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**Syntax**

$$\text{S370FPD}w.d$$

**Syntax Description**

- **w**
  - specifies the width of the input field.
  - Default: 1
  - Range: 1–16

- **d**
  - specifies the power of 10 by which to divide the value. This argument is optional.
  - Default: 0
  - Range: 0–31
Details
Packed decimal data contain two digits per byte, but only one digit in the input field represents the sign. The last half of the last byte indicates the sign: a C or an F for positive numbers and a D for negative numbers.

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

Comparisons
- If you use SAS on an IBM mainframe, the S370FPDw.d and the PDw.d informat 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>

S370FPDUw.d Informat
Reads unsigned packed decimal data in IBM mainframe format.

Category:   Numeric
Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

Syntax
S370FPDUw.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.
Details

Packed decimal data contain two digits per byte. The last half of the last byte, which indicates the sign for signed packed data, is always F for unsigned packed data.

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

Comparisons

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

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>--------1</td>
<td></td>
</tr>
<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 one 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.

- **Category:** Numeric
- **Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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

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 are created in IBM mainframe
format for reading in other operating environments.

Note: Different operating environments store integer binary values in different ways.
This concept is called byte ordering. For a detailed discussion about byte ordering,
see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian
Platforms” on page 231.

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 or Little Endian
  Platforms” on page 232.

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

Example
You can use the INPUT statement and specify the S370FPIB informat. However, this
example uses the informat with the INPUT function, where the binary input value is
described using a hexadecimal literal.

x=input('0100'x,s370fpib2.);

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=4;</td>
<td>256</td>
</tr>
</tbody>
</table>

See Also

Informats:
S370FRBw.d Informat

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

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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:** 6
- **Range:** 2–8

\(d\)

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

- **Range:** 0–10

**Details**

Real binary values are represented in two parts: a mantissa that gives the value, and an exponent that gives the value's magnitude.

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

**Comparisons**

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

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
</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 344

S370FZDw.d Informat

Reads zoned decimal data in IBM mainframe format.

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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 contain decimal points, the d value is ignored. This argument is optional.

Default 0

Range 0–31
Details
Zoned decimal data are similar to standard decimal data in that every digit requires one byte. However, the value's sign is stored in the last byte, along with the last digit.

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

Comparisons
• If you use SAS on an IBM mainframe, S370FZDw.d and ZDw.d are identical.
• The following table shows the equivalent zoned decimal notation for several programming languages:

<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

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 on an IBM mainframe operating environment. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one column of the input field.

See Also

Informats:
• ‘‘ZDw.d Informat’’ on page 389

S370FZDBw.d Informat

Reads zoned decimal data in which zeros have been left blank.

Category: Numeric
Interaction: List input is incompatible with binary input when this informat is specified in a
INFORMAT= statement or an ATTRIB= statement using 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 S370FZDBw.d informat on other operating environments to read zoned decimal
data from IBM mainframe files.

Example
input @1 x s370fzdb8.;

Data Line* | Result
---+-----
| 1
40404040F14040C0 | 1000
40404040F1F2D3 | –123

* The data lines contain a hexadecimal representation of a binary number that is stored in zoned decimal
format on an IBM mainframe operating environment. Two hexadecimal characters correspond to one byte
of binary data, and each byte corresponds to one column of the input field.

S370FZDLw.d Informat
Reads zoned decimal leading-sign data in IBM mainframe format.

Category: Numeric
Interaction: List input is incompatible with binary input when this informat is specified in an `INFORMAT=` statement or an `ATTRIB=` statement using the `INFORMAT=` argument. SAS issues a warning and uses formatted input to read the data.

Syntax

\[ \text{S370FZDL}_{w,d} \]

**Syntax Description**

- \( w \) specifies the width of the input field.
  - Default: 8
  - Range: 1–32

- \( d \) specifies the power of 10 by which to divide the value. This argument is optional.
  - Default: 0
  - Range: 0–31

**Details**

Use `S370FZDL_{w,d}` on other operating environments to read zoned decimal data from IBM mainframe files.

**Comparisons**

- Zoned decimal leading-sign data is similar to standard zoned decimal data except that the sign of the value is stored in the first byte of zoned decimal leading-sign data, along with the first digit.
- The `S370FZDL_{w,d}` informat is equivalent to the COBOL notation PIC S9(\(n\)) DISPLAY SIGN LEADING, where the \(n\) value is the number of digits.

**Example**

```plaintext
input @1 x s370fzd13.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1F2F3</td>
<td>123</td>
</tr>
<tr>
<td>D1F2F3</td>
<td>-123</td>
</tr>
</tbody>
</table>

* The data lines contain a hexadecimal representation of a binary number stored in zoned decimal format on an IBM mainframe operating environment. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one column of the input field.
S370FZDS <w><d> Informat

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

Category: Numeric

Interaction: List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

Syntax

S370FZDS <w><d>

Syntax Description

<w> specifies the width of the input field.

Default: 8

Range: 2–32

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

Default: 0

Range: 0–31

Details

Use S370FZDS <w><d> on other operating environments to read zoned decimal data from IBM mainframe files.

Comparisons

• Zoned decimal separate leading-sign data is similar to standard zoned decimal data except that the sign of the value is stored in the first byte of zoned decimal leading sign data, and the first digit of the value is stored in the second byte.

• The S370FZDS <w><d> informat is equivalent to the COBOL notation PIC S9(n) DISPLAY SIGN LEADING SEPARATE, where the n value is the number of digits.

Example

input @1 x s370fzs4.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>4EF1F2F3</td>
<td>123</td>
</tr>
</tbody>
</table>

---
The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal format on an IBM mainframe operating environment. Each two hexadecimal characters correspond to one 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.

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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 on other operating environments to read zoned decimal data from IBM mainframe files.

**Comparisons**

- Zoned decimal separate trailing-sign data are similar to zoned decimal separate leading-sign data except that the sign of the value is stored in the last byte of zoned decimal separate trailing-sign data.

- The S370FZDTw.d informat is equivalent to the COBOL notation PIC S9(n) DISPLAY SIGN TRAILING SEPARATE, where the \( n \) value is the number of digits.
Example

```plaintext
input @1 x s370fzdt4.;
```

<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 on an IBM mainframe operating environment. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one column of the input field.

S370FZDUw.d Informat

Reads unsigned zoned decimal data in IBM mainframe format.

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**Syntax**

```
S370FZDUw.d
```

**Syntax Description**

- \textbf{w} specifies the width of the input field.
  - Default: 8
  - Range: 1–32

- \textbf{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 on 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 the n value 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 on an IBM mainframe operating environment. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one column of the input field.

**SHRSTAMPw. Informat**

Reads date and time values of SHR records.

**Category:** Date and Time

**Syntax**

`SHRSTAMPw.`

**Syntax Description**

`w`

- specifies the width of the input field.

**Requirement**

- w must be 8 because packed decimal date and time values in SHR records contain eight bytes of information: four bytes of date data that are followed by four bytes of time data.

**Details**

The SHRSTAMPw. informat reads packed decimal date and time values of SHR records that are produced by IBM mainframe environments and converts the date and time values to SAS datetime values.

The general form of the date and time information in an SHR record in hexadecimal notation is `ccyydddFhhmmsssth`, where

- `ccyy` is the two-byte representation of the year. The `cc` portion is the one-byte representation of a two-digit integer that represents the century. The `yy` portion is the one-byte representation of two digits that correspond to the year.
The *cc* portion is the century indicator where 00 indicates 19yy, 01 indicates 20yy, 02 indicates 21yy, and so on. A hexadecimal year value of 0115 is equal to the year 2015.

*ddd* is the one-and-a-half bytes that contain three digits that correspond to the day of the year.

*F* is the half-byte that contains all binary 1s.

*hh* is the one-byte representation of two digits that correspond to the hour of the day.

*mm* is the one-byte representation of two digits that correspond to minutes.

*ss* is the one-byte representation of two digits that correspond to seconds.

*th* is the one-byte representation of two digits that correspond to a 100th of a second.

The SHRSTAMP*w* informat enables you to read, on any operation environment, packed decimal date and time values from files that are created on an IBM mainframe.

**Example**

```
input begin: $hex16.;
y=input(begin, shrstamp8.);
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>----- -----</td>
<td>--------------</td>
</tr>
<tr>
<td>0110239F12403576</td>
<td>1598532035.8</td>
</tr>
</tbody>
</table>

* The data line is a hexadecimal representation of a packed decimal date and time value that is stored as it would appear in an SHR record. Each byte occupies one column of the input field. The result is a SAS datetime value that corresponds to August 27, 2010 12:40:36.

---

**SMFSTAMPw. Informat**

Reads time and date values of SMF records.

**Category:** Date and Time

**Syntax**

`SMFSTAMPw.`

**Syntax Description**

`w` specifies the width of the input field.
Requirement

Requirement w must be 8 because time and date values in SMF records contain eight bytes of information: four bytes of time data that are followed by four bytes of date data.

Tip

Tip The time portion of an SMF record is a four-byte integer binary number that represents time as the number of hundredths of a second past midnight.

Details

The SMFSTAMPw. informat reads integer binary time values and packed decimal date values of SMF records that are produced by IBM mainframe systems and converts the time and date values to SAS datetime values.

The date portion of an SMF record in hexadecimal notation is ccyydddF:

- cc is the one-byte representation of two digits that correspond to the century.
- yy is the one-byte representation of two digits that correspond to the year.
- ddd is the one-and-a-half bytes that contain three digits that correspond to the day of the year.
- F is the half byte that contains all binary 1s.

The SMFSTAMPw. informat enables you to read, on any operating environment, integer binary time values and packed decimal date values from files that are created on an IBM mainframe.

Example

```
input begin: $hex16.;
y=input(begin, smfstamp8.);
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0058DC0C0108200F</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, 2008, 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
Syntax
STIMERw.

Syntax Description
w
  specifies the width of the input field.

Details
The STIMER informat reads performance statistics that the STIMER system option
writes to the SAS log.

The informat reads time values and determines whether the values are hours, minutes, or
seconds based on the presence of decimal points and colons:
•  If no colon is present, the value is the number of seconds.
•  If a single colon is present, the value before the colon is the number of minutes. The
  value after the colon is the number of seconds.
•  If two colons are present, the sequence of time is hours, minutes, and then seconds.
In all cases, the result is a SAS time value.

The input values for STIMER must be in one of the following forms:
•  ss
•  ss.ss
•  mm:ss
•  mm:ss.ss
•  hh:mm:ss
•  hh:mm:ss.ss
  ss
  is an integer that represents the number of seconds.

  mm
  is an integer that represents the number of minutes.

  hh
  is an integer that represents the number of hours.

TIMEw. Informat
Reads hours, minutes, and seconds in the form hh:mm:ss.ss, where special characters such as the colon
(:) or the period (.) are used to separate the hours, minutes, and seconds.

Category:  Date and Time

Syntax
TIMEw.
**Syntax Description**

\[ w \]

Specifies the width of the input field.

- Default: 8
- Range: 5–32

**Details**

The `TIMEw.` informat reads SAS time values in the form: `hh:mm:ss<.ss> <AM | PM>`:

- `hh` is an integer that represents the number of hours.
- `:` represents a special character that separates hours, minutes, and seconds.
- `mm` is an integer between 00 and 59 that represents minutes.
- `ss<.ss>` is an integer that represents the number of seconds, and if needed, tenths of a second. Seconds and tenths of a second must always be separated by a period.
- `AM | PM`
  - AM indicates time between 12:00 midnight and 11:59 in the morning. PM indicates time between 12:00 noon and 11:59 at night.

Separate `hh`, `mm`, and `ss` with a special character. When the period is used as the special character, the time is interpreted in the order hours, minutes, and seconds. For example, 23.22 is 23 hours and 22 minutes, not 23 minutes and 22 seconds, or 23 seconds and 22 tenths of a second.

If you do not enter a value for seconds, SAS assumes a value of 0.

The stored value is the total number of seconds in the time value.

**Example**

```sas
input begin time10.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Formatted with TIMEw.</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.56</td>
<td>46560</td>
<td>12:56:00</td>
</tr>
<tr>
<td>120:320</td>
<td>439200</td>
<td>122:00:00</td>
</tr>
<tr>
<td>1:13 pm</td>
<td>47580</td>
<td>13:13:00</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “HHMMw.d Format” on page 126
TODSTAMPw. Informat
Reads an eight-byte time-of-day stamp.

Category: Date and Time

Syntax
TODSTAMPw.

Syntax Description
w
specifies the width of the input field.

Requirement w must be 8 because the OS TIME macro or the STCK instruction on IBM mainframes each return an eight-byte value.

Details
The TODSTAMPw. informat reads time-of-day clock values that are produced by IBM mainframe operating systems and converts the clock values to SAS datetime values.
If the time-of-day value is all 0s, TODSTAMPw. results in a missing value.
Use TODSTAMPw. on other operating environments to read time-of-day values that are produced by an IBM mainframe.

Example
input btime: $hex16.;
y=input(btime, todstamp8.);

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></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 March 21, 2001, 09:41:45.

**TRAILSGNw. Informat**

Reads a trailing plus (+) or minus (–) sign.

**Category:** Numeric

**Syntax**

```
TRAILSGNw.
```

**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</td>
<td>1–32</td>
</tr>
</tbody>
</table>

**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>---</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>
<tr>
<td>1.2</td>
<td>1.2</td>
</tr>
</tbody>
</table>
TUw. Informat

Reads timer units.

Category: Date and Time

Syntax

\[ \text{TUw.} \]

Syntax Description

\[ w \]

specifies the width of the input field.

Requirement \( w \) must be 4 because the OS TIME macro returns a four-byte value.

Details

The 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 on other operating environments.

Example

\[
\text{input btime tu4.;}
\]

Data Line | Result
---|---
1.2+ | 1.2
1.2- | -1.2

The data line is a hexadecimal representation of a binary, four-byte timer unit value. Each byte occupies one column of the input field. The result is a SAS time value that corresponds to 5:26:58.41 p.m.
VAXRBw.d Informat

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

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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 on other operating environments.

**Comparisons**

If you use SAS that is running under VMS, the VAXRBw.d and the RBw.d informats are identical.

**See Also**

- Informats:
  - “RBw.d Informat” on page 344

VMSZNw.d Informat

Reads VMS and MicroFocus COBOL zoned numeric data.

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.
Syntax
VMSZN\(w.d\)

Syntax Description

\(w\)

specifies the width of the output field.

Default 1

Range 1–32

\(d\)

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

This argument is optional.

Details

The VMSZN\(w.d\) informat is similar to the ZD\(w.d\) informat. Both read a string of ASCII digits, and the last digit is a special character denoting the magnitude of the last digit and the sign of the entire number. The difference between the VMSZN\(w.d\) informat and the ZD\(w.d\) informat is in the special character used for the last digit. The following table shows the special characters used by the VMSZN\(w.d\) informat.

<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

input @1 vmszn4.;
### See Also

**Formats:**
- “VMSZNw.d Format” on page 190

**Informats:**
- “ZDw.d Informat” on page 389

---

## w.d Informat

Reads standard numeric data.

**Category:** 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 contain decimal points, the \( d \) value is ignored. This argument is optional.
- Range: 0–31

### Details

The \( w.d \) informat reads numeric values that are located anywhere in the field. Blanks can precede or follow a numeric value with no effect. A minus sign with no separating blank should immediately precede a negative value. The \( w.d \) informat reads values with decimal points and values in scientific E notation, and it interprets a single period as a missing value.
Comparisons

• The \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
input @1 x 6. @10 y 6.2;
put x @7 y;
```

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

\textbf{Category:} Date and Time

\textbf{Syntax}

\texttt{WEEKUw.}

\textbf{Syntax Description}

\texttt{w}

specifies the width of the input field.

\textbf{Default} 11

\textbf{Range} 3–200
Details

The 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 WEEKUw. uses the current year as the year expression, which is the default. If the input does not contain a day expression, then WEEKUw. uses the first day of the week as the day expression, which is the default.

The U Algorithm calculates the SAS date value using the number-of-week value within the year (Sunday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

The inputs to the WEEKUw. informat are the same date for the following example. The current year is 2012.

<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>12W01</td>
</tr>
<tr>
<td>7-8</td>
<td>yyWwdd</td>
<td>12W0101</td>
</tr>
<tr>
<td>9-10</td>
<td>yyyyWwdd</td>
<td>2012W0101</td>
</tr>
<tr>
<td>11-200</td>
<td>yyyy-Www-dd</td>
<td>2012-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 week.

Example

The current year is 2012 in the following examples.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### WEEKVw. Informat

Reads a value in the form a week-number within a year and returns a SAS date value using the V algorithm.

**Category:** Date and Time

### Syntax

```sas
WEEKVw.
```

### Syntax Description

`w`

specifies the width of the input field.

- **Default:** 11
- **Range:** 3–200

---

### See Also

**Formats:**
- “WEEKUw. Format” on page 197
- “WEEKVw. Format” on page 199
- “WEEKWw. Format” on page 200

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

**Informats:**
- “WEEKVw. Informat” on page 379
- “WEEKWw. Informat” on page 381

---

```sas
v=input('W01',weeku3.);
w=input('03W01',weeku5.);
x=input('03W0101',weeku7.);
y=input('2003W0101',weeku9.);
z=input('2003-W01-01',weeku11.);
put v;
put w;
put x;
put y;
put z;
```

<table>
<thead>
<tr>
<th>Statements</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>v=input('W01',weeku3.);</td>
<td></td>
</tr>
<tr>
<td>w=input('03W01',weeku5.);</td>
<td></td>
</tr>
<tr>
<td>x=input('03W0101',weeku7.);</td>
<td></td>
</tr>
<tr>
<td>y=input('2003W0101',weeku9.);</td>
<td></td>
</tr>
<tr>
<td>z=input('2003-W01-01',weeku11.);</td>
<td></td>
</tr>
<tr>
<td>put v;</td>
<td>18993</td>
</tr>
<tr>
<td>put w;</td>
<td>18993</td>
</tr>
<tr>
<td>put x;</td>
<td>18993</td>
</tr>
<tr>
<td>put y;</td>
<td>18993</td>
</tr>
<tr>
<td>put z;</td>
<td>18993</td>
</tr>
</tbody>
</table>
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 zero 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 2012.

<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>12W01</td>
</tr>
<tr>
<td>7-8</td>
<td>yyWwwdd</td>
<td>12W0101</td>
</tr>
<tr>
<td>9-10</td>
<td>yyyyWwwdd</td>
<td>2012W0101</td>
</tr>
<tr>
<td>11-200</td>
<td>yyyy-Www-dd</td>
<td>2012-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 week.

Example

The current year is 2012 in the following examples.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- - - - - - 1</td>
</tr>
</tbody>
</table>
WEEKWw. Informat

Reads a value in the form of a week-number within the year and returns a SAS date value using the W algorithm.

**Category:** Date and Time

**Syntax**

\texttt{WEEKWw.}

**Syntax Description**

\texttt{w}

specifies the width of the input field.

**Default** 11

**Range** 3–200

---

See Also

Formats:
- “WEEKUw. Format” on page 197
- “WEEKVw. Format” on page 199
- “WEEKWw. Format” on page 200

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

Informat:
- “WEEKUw. Informat” on page 377
- “WEEKWw. Informat” on page 381

---

```
V=INPUT('W01', 'weekv3.);
W=INPUT('03W01', 'weekv5.);
X=INPUT('03W0101', 'weekv7.);
Y=INPUT('2003W0101', 'weekv9.);
Z=INPUT('2003-W01-01', 'weekv11.);
PUT V;
PUT W; 18994
PUT X; 18994
PUT Y; 18994
PUT Z; 18994
```
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. Algorithm W calculates the SAS date value using the number of the week within the year (Monday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

The inputs to the WEEKWw. informat are the same date for the following example. The current year is 2012.

<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>12W01</td>
</tr>
<tr>
<td>7-8</td>
<td>yyWwwdd</td>
<td>12W0101</td>
</tr>
<tr>
<td>9-10</td>
<td>yyyyWwwdd</td>
<td>2012W0101</td>
</tr>
<tr>
<td>11-200</td>
<td>yyyy-Www-dd</td>
<td>2012-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 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 2012 in the following examples.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>------</td>
</tr>
</tbody>
</table>
v=input('W01',weekw3.);
w=input('03W01',weekw5.);
x=input('03W0101',weekw7.);
y=input('2003W0101',weekw9.);
z=input('2003-W01-01',weekw11.);
put v;
put w;
put x;
put y;
put z;

See Also

Formats:
- “WEEKUw. Format” on page 197
- “WEEKVw. Format” on page 199
- “WEEKWw. Format” on page 200

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

Informats:
- “WEEKUw. Informat” on page 377
- “WEEKVw. Informat” on page 379

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 2 or 4 digits.

Category: Date and Time
Alignment: Right

Syntax

YMDDTTMw.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 YMDDTTMw.d informat reads datetime values with required separators in the form <yy>yy-mm-dd hh:mm:ss.ss.

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.

Example

```plaintext
input @1 dt ymddttm24.;
```
<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-03-16 11:23:07.4</td>
<td>1647516187.4</td>
</tr>
<tr>
<td>2012 03 16 11 23 07.4</td>
<td>1647516187.4</td>
</tr>
<tr>
<td>12.3.16/11:23</td>
<td>1647516180</td>
</tr>
</tbody>
</table>

**See Also**

Informats:
- “DATETIMEw. Informat” on page 301
- “MDYAMPw.d Informat” on page 325

**YYMMDDw. Informat**

Reads date values in the form **yyymdd or yyyyymmdd**.

**Category:** Date and Time

**Syntax**

`YYMMDDw.`

**Syntax Description**

`w` specifies the width of the input field.

- Default 6
- Range 6–32

**Details**

SAS read date values in one of the following forms:
- `yyymdd`
- `yyyyymmdd`

`yy` or `yyyy`

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

`mm`

is an integer between 01 and 12 that represents the month of the year.

`dd`

is an integer between 01 and 31 that represents the day of the month.

You can separate the year, month, and day values by blanks or by special characters. However, if delimiters are used, place them between all the values. You can also place
blanks before and after the date. Make sure the width of the input field allows space for blanks and special characters.

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

Example

    input calendar_date yymdd10.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>120316</td>
<td>19068</td>
</tr>
<tr>
<td>12/03/16</td>
<td>19068</td>
</tr>
<tr>
<td>12 03 16</td>
<td>19068</td>
</tr>
<tr>
<td>2012-03-16</td>
<td>19068</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “DATEw. Format” on page 89
- “DDMMYYw. Format” on page 95
- “MMDDYYw. Format” on page 137
- “YYMMDDw. Format” on page 209

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 300
- “DDMMYYw. Informat” on page 303
- “MMDDYYw. Informat” on page 327

System Options:

- “YEARCUTOFF= System Option” in SAS System Options: Reference
YYMMNw. Informat

Reads date values in the form yyyymm or yymm.

Category: Date and Time

Syntax

YYMMNw.

Syntax Description

w

specifies the width of the input field.

Default 4

Range 4–6

Details

SAS reads date values in one of the following forms:

- yyyymm
- yymm

yy or yyyy

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

mm

is a two-digit integer that represents the month.

The N in the informat name must be used and indicates that you cannot separate the year and month values by blanks or by special characters. SAS automatically adds a day value of 01 to the value to make a valid SAS date variable.

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

Example

    input date1 yymmn6.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>--------</td>
</tr>
<tr>
<td>201208</td>
<td>19206</td>
</tr>
</tbody>
</table>

See Also

Formats:
YYQw. Informat

Reads quarters of the year in the form yyQq or yyyyQq.

**Syntax**

```
YYQw.
```

**Syntax Description**

`w`

specifies the width of the input field.

**Default**

6 (For SAS version 6, the default is 4.)

**Range**

4–32 (For SAS version 6, the range is 4–6.)

**Details**

SAS reads data in one of the following forms:

- `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 belonging to the 100-year span that is defined
by the YEARCUTOFF= system option.

Example

input quarter yyq9.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12Q2</td>
<td>19084</td>
</tr>
<tr>
<td>12Q02</td>
<td>19084</td>
</tr>
<tr>
<td>2012Q02</td>
<td>19084</td>
</tr>
</tbody>
</table>

See Also

Functions:
• “QTR Function” in SAS Functions and CALL Routines: Reference
• “YEAR Function” in SAS Functions and CALL Routines: Reference
• “YYQ Function” in SAS Functions and CALL Routines: Reference

System Options:
• “YEARCUTOFF= System Option” in SAS System Options: Reference

ZDw.d Informat
Reads zoned decimal data.

Category: Numeric
Interaction: List input is incompatible with binary input when this informat is specified in an
INFORMAT= statement or an ATTRIB= statement using the INFORMAT= argument.
SAS issues a warning and uses formatted input to read the data.

See: “ZDw.d Informat: UNIX” in SAS Companion for UNIX Environments
     “ZDw.d Informat: Windows” in SAS Companion for Windows
     “ZDw.d Format: z/OS” in SAS Companion for z/OS
Syntax

\texttt{ZDw.d}

Syntax Description

\textit{w}

specifies the width of the input field.

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

\textit{d}

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

| Range   | 1–31 |

Details

The \texttt{ZDw.d} informat reads zoned decimal data in which every digit requires one byte and in which the last byte contains the value's sign along with the last digit.

\textit{Note:} Different operating environments store zoned decimal values in different ways. However, \texttt{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 \texttt{w.d} informat, the \texttt{ZDw.d} informat reads data in which every digit requires one byte. Use \texttt{ZDVw.d} or \texttt{ZDw.d} to read zoned decimal data in which the last byte contains the last digit and the sign.
- The \texttt{ZDw.d} informat functions like the \texttt{ZDVw.d} informat with one exception: \texttt{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>\texttt{ZD3.}</td>
</tr>
<tr>
<td>PL/I</td>
<td>\texttt{PICTURE'99T'}</td>
</tr>
<tr>
<td>COBOL</td>
<td>\texttt{DISPLAY PIC S 999}</td>
</tr>
<tr>
<td>IBM assembler</td>
<td>\texttt{ZL3}</td>
</tr>
</tbody>
</table>
Example

input @1 x zdd.;

<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 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 376
- “ZDVw.d Informat” on page 392

ZDBw.d Informat

Reads zoned decimal data in which zeros have been left blank.

**Category:** Numeric

**Interaction:** List input is incompatible with binary input when this informat is specified in an INFORMAT= statement or an ATTRIB= statement using 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

ZDBw.d

**Syntax Description**

\( w \)

specifies the width of the input field.

**Default** 1

**Range** 1–32

\( d \)

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

**Range** 0–31

Details

The ZDBw.d informat reads zoned decimal data that are produced in IBM 1410, 1401, and 1620 form, where 0s are left blank rather than being punched.
Example

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

<table>
<thead>
<tr>
<th>Data Line *</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------1</td>
<td></td>
</tr>
<tr>
<td>F140C2</td>
<td>102</td>
</tr>
</tbody>
</table>

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal form, including the codes for spaces, on an IBM mainframe operating environment. Each byte occupies one column of the input field.

**ZDVw.d Informat**

Reads and validates zoned decimal data.

**Category:** 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 using the INFORMAT= argument. SAS issues a warning and uses formatted input to read the data.

**Syntax**

```
ZDVw.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 one byte and in which the last byte contains the value's sign along with the last digit. It also validates the input string and disallows invalid data.

ZDVw.d is dependent on the operating environment. For example, on IBM mainframes, 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 ranging from A-F, where A, C, E, and F are positive values and B and D are negative values. The low-
order nibble on IBM mainframes must be a numeric digit that ranges from 0-9, as with ZD.

Note: Different operating environments store zoned decimal values in different ways. However, the ZDVw.d informat reads zoned decimal values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

Comparisons
The ZDVw.d informat functions like the ZDw.d informat with one exception: ZDVw.d validates the input string and disallows invalid data.

Example

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

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>F0F1F2C8</td>
<td>128</td>
</tr>
</tbody>
</table>

* The data line contains a hexadecimal representation of a binary number stored in zoned decimal form. The example was run on an IBM mainframe. The results might vary depending on your operating environment.

See Also

Informats:
- “w.d Informat” on page 376
- “ZDw.d Informat” on page 389
Recommended Reading

Here is the recommended reading list for this title:

- *An Array of Challenges--Test Your SAS Skills*
- *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](http://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](mailto:sasbook@sas.com)
Web address: [sas.com/store/books](http://sas.com/store/books)
## Special Characters

<table>
<thead>
<tr>
<th>Format</th>
<th>Informats</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ASCIIw. format</td>
<td>42</td>
</tr>
<tr>
<td>$ASCIIw. informat</td>
<td>254</td>
</tr>
<tr>
<td>$BASE64Xw. format</td>
<td>43</td>
</tr>
<tr>
<td>$BASE64Xw. informat</td>
<td>255</td>
</tr>
<tr>
<td>$BINARYw. format</td>
<td>44</td>
</tr>
<tr>
<td>$BINARYw. informat</td>
<td>256</td>
</tr>
<tr>
<td>$SCBw. informat</td>
<td>257</td>
</tr>
<tr>
<td>$SCHARw. format</td>
<td>45</td>
</tr>
<tr>
<td>$SCHARw. informat</td>
<td>258</td>
</tr>
</tbody>
</table>

Compared to $ASCII informat 254 compared to $SCHARZBw. informat 260 compared to $SEBCDICw. informat 261 compared to $S. informat 273

<table>
<thead>
<tr>
<th>Format</th>
<th>Informats</th>
</tr>
</thead>
<tbody>
<tr>
<td>$SEBCDICw. format</td>
<td>46</td>
</tr>
<tr>
<td>$SEBCDICw. informat</td>
<td>260</td>
</tr>
</tbody>
</table>

Compared to $S8601CI informat 282 $SN8601Aw.d format 50 $SN8601Aw.d informat 263 $SN8601Bw.d format 49 $SN8601Bw.d informat 263 $SN8601Cw.d format 53 $SN8601Cw.d informat 266 $SN8601Dw.d format 54 $SN8601Dw.d informat 266 $SN8601Exw.d format 52 $SN8601Exw.d informat 265 $SN8601Exw.d informat 265 $SN8601Tw.d format 57 $SN8601Tw.d format 57 $SN8601Wd.d format 58 $SN8601Wd.d format 58 $SOCTALw. format | 59 |
| $SOCTALw. format | 267 |
| $SPHEXw. format | 268 |
| $QUOTEw. format | 60 |
| $QUOTEw. informat | 269 |
| $REVERJw. format | 62 |
| $REVERSw. format | 63 |
| $SUPCASEw. format | 64 |
| $SUPCASEw. informat | 270 |

### %SYSFUNC function

Specifying formats with 5

## A

AM or PM
datetime values with 90, 325 time values with 186

ANYDTDEw. informat 273 ANYDTDMw. informat 276 ANYDTTMw. informat 279 ASCII

Converting character data to 42 ASCII data

Converting character data to, Base 64 encoding 43, 255

Converting to native format 254

ATTRIB statement

Specifying formats with 6 Specifying informats with 230

## B

B8601CI informat 282 B8601DAw. format 71 B8601DAw. informat 283 B8601DJ informat 284 B8601DNw. format 72 B8601DNw. informat 286 B8601DTw.d format 73 B8601DTw.d informat 287 B8601DXw. format 75 B8601DXw. informat

See B8601DZw.d informat

B8601DXw.d informat 289 B8601Dw. format 76 B8601Dz.d informat 289 B8601LXw. format 78 B8601LXw. informat
See B8601DTw.d informat
B8601LXw.d informat 287
B8601LZw. informat 79
B8601TMw.d format 81
B8601TMw.d informat 291
B8601TXw. format 82
B8601TXw. informat
See B8601TZw.d informat
B8601TXw.d informat 292
B8601TZw.d format 83
B8601TZw.d informat 292
Base 64 encoding
converting character data to ASCII text 43, 255
BASE64X 255
BESTw.p format 69
BESTw. format 67
BESTw.d informat 376
big endian platforms
byte ordering 8
big endian platforms, byte ordering on 231
binary
converting character data to 44
converting numeric values to 71
binary data, converting to character 256
integers 294
binary zeros, converting to blanks 260
BINARYw. format 71
BINARYw.d informat 294
bits, extracting 295
BITSw.d informat 295
blanks
converting binary zeros to 260
converting to zeros 296
byte ordering 8, 231
BZw.d informat 296
compared to w.d informat 377

C
CBw.d informat 297
character data
converting to ASCII 42
converting to ASCII text, Base 64 encoding 43, 255
converting to binary 44
converting to EBCDIC 46
converting to hexadecimal 47
converting to octal 59
reverse order, left alignment 63
reverse order, preserving blanks 62
uppercase conversion 64
varying length 64
writing 45, 66
writing in uppercase 48
character data, reading from column-binary files 257
standard format 272
varying length fields 270
with blanks 257
column-binary data, reading down a column 348
punch-card code 343
column-binary files, reading 257
column-binary, reading with blanks 258
commas
in numeric values 85, 86
replacing decimal points with 149
COMMAw.d format 85
COMMAw.d informat 298
compared to COMMAXw.d informat 300
COMMAXw.d format 86
COMMAXw.d informat 299
compared to COMMAw.d informat 299

D
data conversion
formats and 9
data values, reading 227
date and time informats
B8601DN informat, ISO 8601 basic date notation, returns the date in a datetime value 286
B8601DT informat, ISO 8601 basic datetime notation, no time zone 287
reading IBM dates and times 282
reading Java dates and times 284
date and time values
SHR records 366
date values
as day of month 94
B8601DA format, ISO 8601 basic notation 71
B8601DA informat, ISO 8601 basic notation 283
DATEw. format 89
day-of-week name 101
DDMYYw. format 95
DDMYYxw. format 96
DTDAtew. format 102
E8601DA format, ISO 8601 extended notation 108
E8601DA informat, extended notation 305
E8601DN informat, ISO 8601 extended notation, returns date in datetime value 306
extracting from informat values 273
Julian dates 134
Julian day of the year 133
MMDDYYw. format 137
MMDDYYxw. format 139
MMYYw. format 142
MMYYxw. format 143
month name 145
month of the year 146
MONYYw. format 147
quarter of the year 164
quarter of the year in Roman numerals 165
WEEKDATEw. format 193
WEEKDATXw. format 195
WEEKDAYw. format 196
WORDDATEw. format 202
WORDDATXw. format 210
YYMMw. format 207
YYMMxw. format 214
YYMONw. format 214
YYQRw. format 218
YYRwx. format 219
YYQw. format 215
YYQxw. format 216
date/time values, reading
date values, dddmmmyy 300
date values, dddmmmyy hh:mm:ss 301
date values, dddmmmyyyy 300
date values, dddmmmyyyyy hh:mm:ss 301
date values, dmmmyy 303
date values, dmmmyyyy 303
date, yymm 387
date, yymmm 387
dates, mmddyy 327
dates, mmddyyyy 327
dates, yymmmd8 385
dates, yyyyymmdd 385
IBM mainframes 337
IBM mainframes, RMF records 347
IBM mainframes, SMF records 367
Julian dates 324
month and year values 328
RMF records 337
SMF records 337
TIME MIC values 330
time values, IBM mainframe 330
time-of-day stamp 371
time, hh:mm:ss 369
timer units 373
year quarter 388
DATEAMPMw.d format 90
datetime values
$N8601E format, ISO 8601 extended notation 53
datetime formats
ISO 8601 extended datetime with no time zone 110
ISO 8601 extended datetime, with time zone 113
datetime informats
B8601DZ informat, ISO 8601 basic notation with time zone 289
datetime values
$N8601 informat, ISO 8601 basic and extended notation 263
$N8601B format, basic notation 49
$N8601BA format, ISO 8601 basic notation 50
$N8601E format, extended notation 52
$N8601E informat, extended notation 265
$N8601EH format, ISO 8601 extended notation, hyphen for omitted components 54
$N8601EX format, extended notation, x for omitted components 56
$N8601H format, basic notation, hyphen for omitted components 57
$N8601X format, x for omitted components 58
B8601DN format, ISO 8601 basic datetime notation, formats the date 72
B8601DT format, ISO 8601 basic notation, no time zone 73, 75, 78
B8601DZ format, ISO 8601 basic notation with time zone 76
B8601TX format, ISO 8601 basic notation, time zone offset 82
DATEAMPMw.d format 90
DATETIMEw.d format 92
DTDATEw. format 102
DTMONNYw. format 103
DTWKDATXw. format 104
DTYEARw. format 106
DTYYQCw. format 107
E8601DN format, ISO 8601 extended, formats the date 109
E8601DT format, ISO 8601 basic notation, no time zone 112, 115
E8601DT informat, ISO 8601 extended notation, no time zone 307
E8601DZ informat, ISO 8601 extended notation with time zone 309
E8601LZ informat, ISO 8601 extended local notation with time zone 311
E8601TX format, ISO 8601 basic notation, time zone offset 119
extracting from informat values 276
with AM or PM 90
datetime values
YMDDTTMw.d informat 383
DATETIMEw. informat 301
DATETIMEw.d format 92
DATEw. format 89
DATEw. informat 300
DAYw. format 94
DDMMYYw. format 95
DDMMYYw. informat 303
DDMMYYXw. format 96
DEC format
integer binary (fixed-point) values in 131
positive integer binary (fixed-point) values in 160
reading integer binary values in 321
reading positive integer binary values in 341
decimal places
aligned 69, 87
decimal points
replacing with commas 149
decimal points, reading as commas 331
DOLLARw.d format 98
DOLLARw.d informat 298
DOLLARXw.d format 100
DOLLARXw.d informat 299
double quotation marks
data values in 60
DOWNAMEw. format 101
DTDATEw. format 102
DTMONYYw. format 103
DTWKDATXw. format 104
DTYEARw. format 106
DTYYQCw. format 107
duration values
$N8601 informt, ISO 8601 basic and extended notation 263
duration values
$N8601B format, basic notation 49
$N8601BA format, ISO 8601 basic notation 50
$N8601E format, extended notation 52
$N8601E informat, extended notation 265
$N8601EA format, ISO 8601 extended notation 53
$N8601EH format, ISO 8601 extended notation, hyphen for omitted components 54
$N8601EX formats, extended notation, x for omitted components 56
$N8601H format, basic notation, hyphen for omitted components 57
$N8601X format, x for omitted components 58
Dw.d informat 376
Dw.p format 87

E
E8601DAw. format 108
E8601DAw. informat 305
E8601DNw. format 109
E8601DNw. informat 306
E8601DTw.d format 110
E8601DTw.d informat 307
E8601DXw. format 112
E8601DXw. informat 309

See E8601DZw.d informat
E8601DXw.d format 309
E8601DXw. format 113
E8601Dz.w.d informat 309
E8601LXw. format 115
E8601LXw. informat

See E8601DTw.d informat
E8601LXw.d format 307
E8601LXw. format 116
E8601LZw.d format 311
E8601TMw.d format 118
E8601TMw.d informat 312
E8601TXw. format 119
E8601TXw. informat

See E8601TZw.d informat
E8601TXw.d format 314
E8601TZw.d format 121
E8601TZw.d informat 314
EBCDIC
converting character data to numeric data in 168
EBCDIC data
convert to native format 261
reading 350
encoded characters, removing 298, 299
encoding formats and 9
Ew. format 108
Ew.d informat 376

F
fixed-point values
DEC format 131, 160
Intel format 131, 160
reading in Intel and DEC formats 321, 341
writing 129, 158
floating-point data (IEEE), reading 322
floating-point data, reading 316
floating-point values 122
IEEE 132
FLOATw.d format 122
FLOATw.d informat 316
FORMAT statement specifying formats with 6
formats 4
byte ordering and 8
data conversions 9
encodings 9
integer binary notation 9
packed decimal data 10
permanent 6
specifying with %SYSFUNC function 5
specifying with ATTRIB statement 6
specifying with FORMAT statement 6
specifying with PUT function 5
specifying with PUT statement 5
syntax 4
temporary 6
user-defined 7
zoned decimal data 10
formatting symbols
ISO 8601 14, 237
fractions 124, 204
FRACTw. format 124
Fw.d format 192
Fw.d informat 376

**H**

HEX 47
hexadecimal converting character data to 47
converting real binary (floating-point) values to 125
packed Julian dates in 153
packed Julian dates in, for IBM 154
hexadecimal binary values, converting to integers 317
hexadecimal binary values, converting to real binary 317
hexadecimal data, converting to character 262
hexadecimal values reading packed Julian date values in, for IBM 334
reading packed Julian dates in, for IBM 336
HEXw. format 125
HEXw. informat 317
compared to $HEXw. informat 262
HHMMSSw. informat 318
HHMMw.d format 126
HOURw.d format 128

**I**

IBM packed Julian dates in hexadecimal for 154
IBM dates and times, reading 282, 284
IBM mainframe format integer binary (fixed-point) values in 169
numeric data in 168
packed decimal data in 172
positive integer binary (fixed-point) values in 174
real binary (floating-point) data in 176
unsigned integer binary (fixed-point) values in 170
unsigned packed decimal data in 173
unsigned zoned decimal data in 182
zoned decimal data 177
zoned decimal leading-sign data in 178
zoned decimal separate leading-sign data in 179
zoned decimal separate trailing-sign data in 180
IBM packed decimal data, reading 333
IBRw.d format 131
IBRw.d informat 321
IBw.d format 129
IBw.d informat 320
compared to $370FIBw.d informat 352
IEEE floating-point values 132
reading 322
IEEEw.d format 132
IEEEw.d informat 322
INFORMAT statement specifying informats with 229
informats 227
byte ordering 231
integer binary notation 232
packed decimal data and 13
permanent 230
specifying, with ATTRIB statement 230
specifying, with INFORMAT statement 229
specifying, with INPUT function 229
specifying, with INPUT statement 229
syntax 228
temporary 230
user-defined 230
zoned decimal data and 13
INPUT function
  specifying informats with 229
INPUT statement
  specifying informats with 229
integer binary (fixed-point) values
  IBM mainframe format 169
integer binary data
  byte ordering 8
  notation and programming languages 9
integer binary data, reading
  IBM mainframe format 351, 356
integer binary notation 232
integer binary values
  DEC format 131
  Intel format 131
  reading in Intel and DEC formats 321
writing 129
integer binary values, reading 320, 339
integers
  printing without decimals 69
Intel format
  integer binary (fixed-point) values in 131
  positive integer binary (fixed-point) values in 160
  reading integer binary values in 321
  reading positive integer binary values in 341
interval values
  $N8601BA format, ISO 8601 basic notation 50
  $N8601EA format, ISO 8601 extended notation 53
  $N8601EX formats, extended notation, x for omitted components 56
  $N8601H format, basic notation, hyphen for omitted components 57
interval values
  $N8601 informat, ISO 8601 basic and extended notation 263
  $N8601B format, basic notation 49
  $N8601E format, extended notation 52
  $N8601E informat, extended notation 265
  $N8601EH format, ISO 8601 extended notation, hyphen for omitted components 54
  $N8601X format, x for omitted components 58
IS8601DAw. format 108
IS8601DAw. informat 305
IS8601DNw. informat 306
IS8601DTw.d format 110
IS8601DTw.d informat 307
IS8601DZw. format 113
IS8601DZw.d informat 309
IS8601LZw. format 116
IS8601LZw.d informat 311
IS8601TMw.d format 118
IS8601TMw.d informat 312
IS8601TZw.d format 121
IS8601TZw.d informat 314
ISO 8601 date and time formats
  B8601DA format, basic date notation 71
  B8601DN format, basic datetime notation, formats the date 72
  B8601DT format, basic datetime notation, no time zone 73
  B8601DX format, basic datetime notation, time zone offset 75
  B8601DZ format, basic datetime notation with time zone 76
  B8601LX format, basic datetime notation, time zone offset 78
  B8601LZ format, basic local time with time zone 79
  B8601TM format, basic time notation, no time zone 81
  B8601TX format, basic time notation, time zone offset 82
  B8601TZ format, basic time notation with time zone 83
  E8601DA format, extended date notation 108
  E8601DN format, extended datetime notation, formats the date 109
  E8601DX format, basic datetime notation, time zone offset 112
  E8601LX format, basic datetime notation, time zone offset 115
  E8601TM format, extended time notation, no time zone 118
  E8601TX format, basic time notation, time zone offset 119
  E8601TZ format, extended time notation with time zone 121
  extended datetime, with time zone 113
  extended local time with UTC offset 116
ISO 8601 date and time informats
  $N8601 informat, basic and extended notation for durations, datetimes, and intervals 263
  $N8601E informat, extended notation for duration, datetime, and interval 265
  B8601DA informat, basic date notation 283
B8601DN informat, basic datetime notation, returns the date in a datetime value 286
B8601DZ informat, basic datetime notation with time zone 289
B8601TM informat, basic time notation, no time zone 291
B8601TZ informat, basic time notation with time zone 292
E8601DA informat, extended date notation 305
E8601DN informat, extended notation, returns date in datetime value 306
E8601DT informat, basic datetime notation, no time zone 287
E8601DT informat, extended datetime notation, no time zone 307
E8601DZ informat, extended datetime notation with time zone 309
E8601ILZ informat, extended local datetime notation with time zone 311
E8601TM informat, extended time notation, no time zone 312
E8601TZ informat, extended time notation with time zone 314
ISO 8601 datetime formats
extended datetime with no time zone 110
ISO 8601 duration and datetime formats
$N8601BA format, basic notation 49
$N8601EA format, extended notation 52
$N8601E format, extended notation 53
$N8601EH format, extended notation, hyphen for omitted components 54
$N8601H format, hyphen for omitted components 57
$N8601X format, x for omitted components 58
ISO 8601 formatting symbols 14, 237
ISO 8602 duration and datetime formats
$N8601EX formats, extended notation, x for omitted components 56
packed values in hexadecimal 153
packed values in hexadecimal for IBM 154
Julian dates, packed
reading in hexadecimal format, for IBM 336
JULIANw. format 134
JULIANw. informat 324

L
leading zeros 221
little endian platforms
byte ordering 8
little endian platforms, byte ordering on 231

M
MDYAMPMw.d format 135
MDYAMPMw.d informat 325
MicroFocus COBOL
zoned numeric data 374
MicroFocus Cobol zoned numeric data 190
minus sign
trailing 372
MMDDYYw. format 137
MMDDYYw. informat 327
MMDDYYxw. format 139
MMSSw.d format 141
MMYYw. format 142
MMYYxw. format 143
MONNAMEw. format 145
MONTHw. format 146
MONNYw. format 147
MONYYw. informat 328
MSECw. informat 330

N
N8601B 49
N8601E 52
N8601EH 54
ND8601DAw. informat 283
ND8601DNw. informat 286
ND8601DTw.d informat 287
ND8601DZw.d informat 289
ND8601TMw.d informat 291
ND8601TZw.d informat 292
negative numeric values
writing in parentheses 148
NEGPARENw.d format 148
nibble 233
definition 10
numeric data

J
Java dates and times, reading 284
JULDAYw. format 133
Julian date values, packed
reading in hexadecimal form, for IBM 334
Julian dates 134, 234
day of the year 133
packed 11
commas replacing decimal points 149
EBCDIC format 168
IBM mainframe format 168
leading zeros with 221
one digit per byte 192
scientific notation 108
zoned decimal format 222
numeric data, reading
commas for decimal points 331
from column-binary files 297
standard format 376
numeric values
aligning decimal places 69, 87
best notation 67
commas in 85, 86
converting to binary 71
converting to fractions 124
converting to octal 151
DOLLARw.d format 98
DOLLARXw.d format 100
Roman numerals 167
words with numeric fractions 204
writing as percentages 156
writing as words 205
writing negative values in parentheses 148
NUMXw.d format 149
NUMXw.d informat 331

O
octal
  converting character data to 59
  converting numeric values to 151
octal data
  converting to character 267
  converting to integers 332
OCTALw. format 151
OCTALw. informat
  compared to $OCTALw. informat 268
OCTALw.d informat 332

P
p-values
  writing 163
packed data, reading in IBM mainframe format 354
packed decimal data 11, 234
  defined 233
  definition 10
  formats and 10
  formats and informats for 236
IBM mainframe format 172
languages supporting 12, 235
platforms supporting 12, 235
summary of formats and informats 13
unsigned format 161
packed decimal format
writing data in 151
packed hexadecimal data, converting to character 268
packed Julian date values
  reading in hexadecimal form, for IBM 334
writing in hexadecimal 153
writing in hexadecimal for IBM 154
packed Julian dates 11, 234
  reading in hexadecimal format, for IBM 336
parentheses
  writing negative numeric values in 148
PDJULGw. format 153
PDJULGw. informat 334
PDJULIw. format 154
PDJULIw. informat 336
PDTIMEw. informat 337
  compared to RMFSTAMPw. informat 348
PDw.d format 151
PDw.d informat 333
  compared to $PHEXw. informat 269
  compared to PKw.d informat 343
  compared to S370FPDw.d informat 355
percentages
  converting to numeric values 338
numeric values as 156
  with minus sign for negative values 157
PERCENTNw.d format 157
PERCENTw.d format 156
PERCENTw.d informat 338
permanent formats 6
PIBRw.d format 160
PIBRw.d informat 341
PIBw.d format 158
PIBw.d informat 339
  compared to S370FPIBw.d informat 357
PKw.d format 161
PKw.d informat 342
plus sign
  trailing 372
PM or AM
datetime values with 90
time values with 186
positive integer binary (fixed-point) values
  IBM mainframe format 174
positive integer binary values
  DEC format 160
  Intel format 160
  reading in Intel and DEC formats 341
writing 158
programming languages
integer binary notation and 9
packed decimal data support 12
zoned decimal data support 12
PUNCH.d informat 343
PUT function
specifying formats with 5
PUT statement
specifying formats with 5
PVALUEw.d format 163

Q
QTRRw. format 165
QTRw. format 164
quotation marks
removing 269

R
RBw.d format 165
RBw.d informat 344
compared to S370FRBw.d informat 358
compared to VAXRBw.d informat 374
reading data values 227
real binary (floating-point) data
IBM mainframe format 176
VMS format 189
real binary (floating-point) values
converting to hexadecimal 125
real binary data
real binary format 165
real binary data, reading 344
IBM mainframe format 358
VMS format 374
real binary format
real binary data (floating-point) in reverse order character data 165
RMF records, reading duration intervals 346
RMFDURw. informat 346
RMFSTAMPw. informat 347
compared to RMFDURw. informat 346
roman numerals 218, 219
Roman numerals 165, 167
ROMANw. format 167
ROWw.d informat 348

S
S370FFw.d format 168
S370FFw.d informat 350
S370FIBUw.d format 170
S370FIBUw.d informat 352
S370FIBw.d format 169
S370FIBw.d informat 351
S370FPDUw.d format 173
S370FPDUw.d informat 355
S370FPDw.d format 172
S370FPDw.d informat 354
compared to S370FPDUw.d informat 356
S370FPIBw.d format 174
S370FPIBw.d informat 356
compared to S370FIBUw.d informat 353
S370FRBw.d format 176
S370FRBw.d informat 358
S370FZDB 360
S370FZDBw.d informat 360
S370FZDLw.d format 178
S370FZDLw.d informat 361
S370FZDSw.d format 179
S370FZDSw.d informat 363
S370FZDTw.d format 180
S370FZDTw.d informat 364
S370FZDUw.d format 182
S370FZDUw.d informat 365
S370FZDw.d format 177
S370FZDw.d informat 359
compared to S370FZDUw.d informat 366
SAS informats 227
scientific notation 108
SHR records
reading data and time values of 366
SHRSTAMPw. informat 366
SMFSTAMPw. informat 367
Social Security numbers 183
SSNw. format 183
STIMERw. informat 368

T
temporary formats 6
time values
B8601LZ format, ISO 8601 basic local time with time zone 79
B8601TM format, ISO 8601 basic time notation, no time zone 81
B8601TM informat, ISO 8601 basic time notation, no time zone 291
B8601TZ format, ISO 8601 basic time notation with time zone 83
B8601TZ informat, ISO 8601 basic time notation with time zone 292
E8601TM format, ISO 8601 extended notation, no time zone 118
E8601TM informat, ISO 8601 extended time notation, no time zone 312
E8601TZ format, ISO 8601 extended notation with time zone 121
E8601TZ informat, ISO 8601 extended notation with time zone 314
extracting from informat values 279
HHMMw.d format 126
HOURw.d format 128
ISO 8601 extended local time with UTC offset 116
MMSSw.d format 141
TIMEAMPw.d format 186
TIMEw.d format 184
TODw.d format 187
TIMEAMPw.d format 186
TIMEw. informat 369
TIMEw.d format 184
TODSTAMPw. informat 371
  compared to MSECw. informat 330
TOD.w.d format 187
trailing plus or minus sign 372
TRAILSGNw. informat 372
transcoding 10
TUw. informat 373

U
unsigned integer binary (fixed-point) values
IBM mainframe format 170
unsigned integer binary data, reading
IBM mainframe format 352
unsigned packed decimal data
IBM mainframe format 173
unsigned packed decimal data, reading 342
IBM mainframe format 355
unsigned packed decimal format 161
unsigned zoned decimal data
IBM mainframe format 182
unsigned zoned decimal data, reading
IBM mainframe format 365
uppercase
$UPCASEw. informat 270
converting character data to 64
reading data as 270
writing character data in 48
user-defined formats 7

V
VAXRBw.d format 189
VAXRBw.d informat 374
VMS
zoned numeric data 190, 374
VMS format
real binary (floating-point) data in 189
VMSZNw.d format 190
VMSZNw.d informat 374

W
w.d format 192
w.d informat 376, 392
  compared to NUMXw.d informat 331
  compared to ZDw.d informat 390
WEEKDATEw. format 193
WEEKDATXw. format 195
WEEKDAYw. format 196
weeks
  number of week, date value, U algorithm 377
  week number, date value, V algorithm 379
  week number, date value, W algorithm 381
  week number, decimal format, U algorithm 197
  week number, decimal format, V algorithm 199
  week number, decimal format, W algorithm 200
WEEKU 197
WEEKUw. format 197
WEEKUw. informat 377
WEEKVw. format 199
WEEKVw. informat 379
WEEKWw. format 200
WEEKWw. informat 381
WORDDATEw. format 202
WORDDATXw. format 203
WORDFw. format 204
words
  writing numeric values as 205
WORDSw. format 205
writing character data 45, 66

Y
YEARw. format 206
YMDDTMw.d format 383
YYMDDw. format 209
YYMDDw. informat 385
YYMDDDxw. format 210
YYMNNw. informat 387
YYMMw. format 207
YYMxw. format 212
YYMONw. format 214
YYQRw. format 218
YYQRxw. format 219
YYQw. format 215
YYQw. informat 388
YYQxw. format 216
Z
ZDBw.d informat 391
ZDVw.d informat 392
See also w.d informat
See also ZDw.d informat
compared to ZDw.d informat 390
ZDw.d format 222
ZDw.d informat 389, 392
compared to ZDVw.d 393
zero
numeric data with leading zeros 221
zeros, binary
converting to blanks 260
zoned decimal data 11, 234
defined 233
definition 10
formats and 10
formats and informats for 236
IBM mainframe format 177
languages supporting 12, 235
platforms supporting 12, 235
summary of formats and informats 13
zoned decimal data, reading 389, 391
IBM mainframe format 359
zoned decimal format 222
zoned decimal leading-sign data
IBM mainframe format 178
zoned decimal leading-sign data, reading
IBM mainframe format 361
zoned decimal separate leading-sign data
IBM mainframe format 179
zoned decimal separate trailing-sign data
IBM mainframe format 180
zoned decimal separate trailing-sign data, reading
IBM mainframe format 363
zoned numeric data
MicroFocus COBOL 190, 374
VMS 190, 374
zoned separate leading-sign data, reading
IBM mainframe format 363
Zw.d format 221
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.