THIS DOCUMENTATION FOR AN EARLY ADOPTER PRODUCT IS A PRELIMINARY DRAFT AND IS PROVIDED BY SAS INSTITUTE INC. ("SAS") ON AN "AS IS" BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE. SAS does not warrant that this documentation is complete, accurate, or similar to that which may be released to the general public, or that any such documentation will be released. The company shall not be liable whatsoever for any damages arising out of the use of this documentation, including any direct, indirect, or consequential damages. SAS reserves the right to alter or abandon use of this documentation at any time.
Early Adopter Software
Part 1

SAS Formats

Chapter 1
Overview of Formats .............................................. 3

Chapter 2
Formats Documented in Other Publications .................... 29

Chapter 3
Dictionary of Formats for SAS and the CAS Server ........... 31

Chapter 4
Dictionary of Formats for SAS Only ............................ 131
Chapter 1
Overview of 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 ............................................................ 10
  Platforms That Support Packed Decimal and Zoned Decimal Data .... 11
  Languages That Support Packed Decimal and Zoned Decimal Data .... 12
  Summary of Packed Decimal and Zoned Decimal Formats and Informats .... 12
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 ................ 16
  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 ........... 24
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.

\text{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.

\text{See}

For information about user-defined formats, see “FORMAT” in SAS Viya Utility Procedures Guide.

\text{w}

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

\text{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 \text{w} and \text{d} values from the format, SAS uses default values. The \text{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 \text{w} value of 10 specifies a maximum of 10 columns for the value. The \text{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\text{w}.\text{d} format. SAS prints asterisks if you do not specify an adequate width. In the following example, the result is x=**.

\[
x=123;
\]
\[
\text{put } x= 2.;
\]
If you use an incompatible format, such as a numeric format to write character values, SAS first attempts to use an analogous format of the other type. If this attempt fails, an error message that describes the problem is displayed in the SAS log.

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

## Using Formats

### Ways to Specify Formats

#### About Specifying Formats
You can 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 Viya 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 Viya 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 Viya 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 Viya 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 Viya 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 Viya 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.
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 SAS Viya Utility 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 SAS Viya Utility 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 BESTw. 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, Macintosh, and Solaris on SPARC. The following platforms are considered little endian: Intel ABI, Linux, OpenVMS on HP Integrity Servers, OpenVMS Integrity.

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 01 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 FF FE, FF FF, and 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>
<tr>
<td>Platform for Which the Data Was Created</td>
<td>Platform That Writes the Data</td>
<td>Signed Integer</td>
<td>Format</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>------------------------------</td>
<td>----------------</td>
<td>----------</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>no</td>
<td>S370FPIB</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>no</td>
<td>PIBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>yes</td>
<td>IB or IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>no</td>
<td>PIB or PIBR</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>yes</td>
<td>S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>no</td>
<td>S370FPIB</td>
</tr>
<tr>
<td>little endian</td>
<td>either</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>either</td>
<td>no</td>
<td>PIBR</td>
</tr>
</tbody>
</table>

**Integer Binary Notation and Different Programming Languages**

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

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

**Data Conversions and Encodings**

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

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

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 Viya 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 Informats</th>
</tr>
</thead>
<tbody>
<tr>
<td>PIC S9(X) PACKED-DECIMAL</td>
<td>S370FPD&lt;sub&gt;w&lt;/sub&gt;.</td>
</tr>
<tr>
<td>PIC 9(X) PACKED-DECIMAL</td>
<td>S370FPDU&lt;sub&gt;W&lt;/sub&gt;.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY</td>
<td>S370FZD&lt;sub&gt;W&lt;/sub&gt;.</td>
</tr>
<tr>
<td>PIC 9(W) DISPLAY</td>
<td>S370FZDU&lt;sub&gt;W&lt;/sub&gt;.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN LEADING</td>
<td>S370FZDL&lt;sub&gt;W&lt;/sub&gt;.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN LEADING SEPARATE</td>
<td>S370FZDS&lt;sub&gt;W&lt;/sub&gt;.</td>
</tr>
<tr>
<td>PIC S9(W) DISPLAY SIGN TRAILING SEPARATE</td>
<td>S370FZDT&lt;sub&gt;W&lt;/sub&gt;.</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 \( \lceil \frac{(x+1)}{2} \rceil \). For PIC 9(X) PACKED-DECIMAL, W is \( \lceil \frac{x}{2} \rceil \). For example, PIC S9(5) PACKED-DECIMAL represents five digits. If a sign is included, six nibbles are needed. **\( \lceil \frac{(5+1)}{2} \rceil \)** 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 (in hex)</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 Informats

SAS uses a group of formats and informats to handle packed and zoned decimal data. The following table lists the type of data representation for these formats and informats. The formats and informats that begin with S370 refer to IBM mainframe representation.
<table>
<thead>
<tr>
<th>Format</th>
<th>Type of Data Representation</th>
<th>Corresponding Informat</th>
<th>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>
<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>Format</td>
<td>Type of Data Representation</td>
<td>Corresponding Informat</td>
<td>Comments</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------</td>
<td>------------------------</td>
<td>----------</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 zero-based year from 1900</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>SMFSTAMP</td>
<td>Input layout is: xxxxxxxxxyyyydddF, where yyyydddF is the packed Julian date; yyyy is a zero-based year from 1900</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>PDTIME</td>
<td>Input layout is: 0hhmmssF</td>
</tr>
<tr>
<td>none</td>
<td>Packed decimal</td>
<td>RMFSTAMP</td>
<td>Input layout is: 0hhmmssFyyyydddF, where yyyydddF is the packed Julian date; yyyy is a zero-based year from 1900</td>
</tr>
</tbody>
</table>

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

**ISO 8601 Formatting Symbols**

The following list explains the formatting symbols that are used to notate the 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

---

**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 Viya 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
- 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:
### Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations

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>yyyyymddd</td>
<td>20120915</td>
<td>B8601DAw. on page 157</td>
</tr>
<tr>
<td>Writes the date from a</td>
<td>yyyyymddd</td>
<td>20120915</td>
<td>B8601DNw. on page 158</td>
</tr>
<tr>
<td>datetime.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>hhmmss&lt;fffffff&gt;</td>
<td>155300322348</td>
<td>B8601TMw.d on page 166</td>
</tr>
<tr>
<td>Zero meridian time and</td>
<td>hhmmss+</td>
<td>–</td>
<td>hhmm</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>hhmmssZ</td>
<td>155300Z</td>
<td></td>
<td>B8601TZw. on page 169</td>
</tr>
<tr>
<td>Zero meridian time that</td>
<td>hhmmss+</td>
<td>–</td>
<td>hhmm</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>Converts time to a user</td>
<td>hhmmss+</td>
<td>–</td>
<td>hhmm</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>yyyyymdddThhmmss&lt;fffffff&gt;</td>
<td>20120915T155300</td>
<td>B8601DTw.d on page 159</td>
</tr>
<tr>
<td>Zero meridian datetime</td>
<td>yyyyymdddThhmmss&lt;fffffff&gt;</td>
<td>20120915T155300–0000</td>
<td>B8601DZw. on page 162</td>
</tr>
<tr>
<td>and 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>yyyyymdddThhmmssZ</td>
<td>20120915T155300Z</td>
<td></td>
<td>B8601DZw. on page 162</td>
</tr>
<tr>
<td>Zero meridian datetime</td>
<td>yyyyymdddThhmmss&lt;fffffff&gt;</td>
<td>20120915T155300+0500</td>
<td>B8601LXw. on page 164</td>
</tr>
<tr>
<td>that uses a time zone</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset for the user</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>local time.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Converts a datetime</td>
<td>yyyyymdddThhmmss&lt;fffffff&gt;</td>
<td>20120915T155300+0500</td>
<td>B8601DXw. on page 161</td>
</tr>
<tr>
<td>value to the user local</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>time by using a time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>zone offset for the user</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>local time.</td>
<td></td>
<td></td>
<td></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-ddThh:mm:ss<ffffff>Z`

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

<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>Date</td>
<td><code>yyyy-mm-dd</code></td>
<td>2012-09-15</td>
<td>E8601DAw. on page 171</td>
</tr>
<tr>
<td>Writes the date from a</td>
<td></td>
<td>2012-09-15</td>
<td>E8601DNw. on page 158</td>
</tr>
<tr>
<td>datetime.</td>
<td><code>yyyy-mm-dd</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td><code>hh:mm:ss&lt;ffffff&gt;</code></td>
<td>15:53:00.322348</td>
<td>E8601TMw.d on page 166</td>
</tr>
<tr>
<td>Zero meridian time and</td>
<td><code>hh:mm:ss&lt;ffffff&gt;</code></td>
<td>15:53:00+00:00</td>
<td>E8601TZw.d on page 169</td>
</tr>
<tr>
<td>time zone offset.</td>
<td><code>hh:mm:ss&lt;ffffff&gt;</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td>The time zone offset is</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>always +00:00.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero meridian time that</td>
<td><code>hh:mm:ss&lt;ffffff&gt;</code></td>
<td>15:53:00+05:00</td>
<td>E8601LZw.d on page 165</td>
</tr>
<tr>
<td>uses a local time zone</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Converts time to user</td>
<td>`hh:mm:ss+</td>
<td>-hh:mm`</td>
<td>15:53:00+05:00</td>
</tr>
<tr>
<td>local time by using a</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>local time zone offset.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Datetime</td>
<td><code>yyyy-mm-ddThh:mm:ss&lt;ffffff&gt;</code></td>
<td>2012-09-15T15:53:00</td>
<td>E8601DTw.d on page 159</td>
</tr>
<tr>
<td>Zero meridian datetime</td>
<td><code>yyyy-mm-ddThh:mm:ss&lt;ffffff&gt;</code></td>
<td>2013-09-15T15:53:00+00:00</td>
<td>E8601DZw.d on page 176</td>
</tr>
<tr>
<td>and 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 +00:00.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero meridian datetime</td>
<td><code>yyyy-mm-ddThh:mm:ss&lt;ffffff&gt;</code></td>
<td>2013-09-15T15:53:00–05:00</td>
<td>E8601LZw.d on page 177</td>
</tr>
<tr>
<td>that uses a time zone</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>offset for the user</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>local time.</td>
<td></td>
<td></td>
<td></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>
<tr>
<td>Zero meridian datetime that uses a user local time zone offset</td>
<td></td>
<td></td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>User local time or datetime that uses a user local time zone offset</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>
Examples of Reading and Writing Basic and Extended ISO 8601 Date, Time, and Datetime Values

About the Basic and Extended ISO 8601 Examples
The examples in this section demonstrate how to use various informats to read date, time, and datetime values into SAS date, time, and datetime variables. The examples also illustrate how to use formats to write these values in a way that is meaningful to users.

Using the Examples on CAS
The examples in the following topics are written to run with the SAS Client. To run an example on CAS, add a CAS statement as the first line of the example:

```
cas casuser;
```

Comparing ISO 8601 Extended Format Output
This example compares the output for the different extended notations for time and datetime values.

```
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 TIMEZONE= 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  
e8601dx= 2013-09-15T00:53:00-05:00  
tm= 05:53:00  
e8601tz=05:53:00+00:00  
e8601lz=05:53:00-05:00  
e8601tx=00:53:00-05:00
```

Here are the results when the TIMEZONE= option is set to America/Los_Angeles:
Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations

• 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.

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:

data _null_; 
  input @1 dt b8601dn.;
  put dt b8601dn.;
datalines;
  20130915T094322
;run;

Here is the output from the SAS log:

    20130915
Reading and Writing Time Values with No Time Zone Offsets

You can read time values that do not have time zone offset values into SAS time values by using the B8601TMw.d and E8601TMw.d informats. The B8601TMw.d and the E8601TMw.d formats write time values that do not have time zone offset values, as shown in this example:

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

Here is the output from the SAS log:

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

Reading and Writing Time Values with Time Zone Offsets

This example uses the E8601TZw.d informat to read a time value that contains a time zone offset. The B8601TZw.d and E8601TZw.d formats write the time values by using a time zone offset for the zero meridian:

```sas
data _null_;
x=input('12:34:56-04:00',e8601tz14.);
p=put x=e8601tz14.;
p=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 B8601TXw. or E8601TXw. 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;
p=put x=e8601tx.;
run;
```

Here is the output from the SAS log:

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

Reading and Writing Time Values with Local Time Zone Offsets

Because time values are scalar, SAS does not typically compute time values based on the time zone of the programmer’s location. One exception to this rule occurs when a SAS time (not a datetime) is computed and then formatted with either the B8601LZw. format or the E8601LZw. format. These two formats query the host code to determine the time zone offset. Then, the current local time and the time zone offset (based on your time zone) are displayed accordingly:

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

Here is the output from the SAS log:

```
22
```
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',e8601dx25.);  
put x=e8601dx25.;
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.

```sas
options timezone='america/anchorage';
data _null_;
t='01Feb2013T12:34:56'dt;
pux 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:

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

Here is the output from the SAS log:

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

You can write a user’s local datetime value by using the time zone offset. Specify the user’s time zone by using the TIMEZONE= system option and the B8601LXw. or E8601LXw. formats. This example writes the datetime for a user by using the time zone name PST (Pacific Time):

```sas
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:

- $P_{yyyy}mmddThhmmss$
- $P_{yyyy-mm-dd}Thh:mm:ss$
- $P_{yyyy-mm-dd}$ is a span of years, months, and days
- $P_{nYnMnDTnHnMnS}$ is the number of years (Y), months (M), days (D), hours (H), minutes (M), and seconds (S)
- $P_{nW}$ specifies the number of weeks

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

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

When you use the $P_{nW}$ notation, $W$ must be the only component in the duration.

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

- $datetime/datetime$
- $datetime/duration$
- $duration/datetime$

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>$P_{yyyy}mmddThhmmss$</td>
<td>P20120915T155300</td>
<td>$N8601BA$</td>
</tr>
<tr>
<td>Duration - Extended Notation</td>
<td>$P_{yyyy-mm-dd}Thh:mm:ss.fff$</td>
<td>P2012-09-15T15:53:00</td>
<td>$N8601EA$</td>
</tr>
<tr>
<td>Duration - Basic and Extended Notation</td>
<td>$P_{nYnMnDTnHnMnS}$</td>
<td>P2y10m14dT20h13m45s</td>
<td>$N8601B$ $N8601E$</td>
</tr>
<tr>
<td>Interval - Basic Notation</td>
<td>$yyyyYmmddThhmmssfff/yyyyYmmddThhmmssfff$</td>
<td>20120915T155300/20141113T000000</td>
<td>$N8601BA$</td>
</tr>
<tr>
<td></td>
<td>$P_{nYnMnDTnHnMnS}$/ $yyyyYmmddThhmmssfff$</td>
<td>P2y10M14dT20h13m45s/20120915T155300</td>
<td>$N8601B$</td>
</tr>
</tbody>
</table>
### Time Component

<table>
<thead>
<tr>
<th>ISO 8601 Notation</th>
<th>Example</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>yyyymmddThhmmssfff</code></td>
<td>20120915T155300/P2y10M14d12h13m45s</td>
<td>$N8601BA</td>
</tr>
<tr>
<td><code>PnYnMnDTnHnMnS</code></td>
<td>2012-09-15T15:53:00/2014-11-13T00:00:00</td>
<td>$N8601EA</td>
</tr>
<tr>
<td><code>yyyymm-ddTthh:mm:ssfff</code></td>
<td>P2y10M14d12h13m45s/2012-09-15T15:53:00</td>
<td>$N8601E</td>
</tr>
<tr>
<td><code>yyyymmddThhmmssfff+/–hhmm</code></td>
<td>20120915T155300</td>
<td>$N8601BA</td>
</tr>
<tr>
<td><code>yyyymm-ddTthh:mm:ssfff+/–hhmm</code></td>
<td>2012-09-15T15:53:00/2012-09-15T15:53:00</td>
<td>$N8601EA</td>
</tr>
<tr>
<td><code>yyyymm-ddTthh:mm:ssfff+/–hhmm</code></td>
<td>2012-09-15T15:53:00/2012-09-15T15:53:00</td>
<td>$N8601EA</td>
</tr>
<tr>
<td><code>yyyymm-ddTthh:mm:ssfff+/–hhmm</code></td>
<td>2012-09-15T15:53:00/04:30</td>
<td>$N8601EA</td>
</tr>
</tbody>
</table>

### Writing Partial and Missing Components

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

Missing components in the durations form \( PnYnMnDTnHnMnS \) are dropped; they do not contain a hyphen or an \( x \). For example, in \( P2mT4H \), the year, day, minutes, and seconds are missing and have been dropped.

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

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

The following formats write omitted components that use the hyphen and the \( x \):
### Datetime and Duration Formats

<table>
<thead>
<tr>
<th>Format</th>
<th>Datetime Form</th>
<th>Duration Form</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N8601H</td>
<td>yyyy-mm-ddThh:mm:ss</td>
<td>PnYnMnDTnHnMnS</td>
<td>--09-15T15::53</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>P2Y2DT4H5M6S/--09-15T15::00</td>
</tr>
<tr>
<td>$N8601EH</td>
<td>yyyy-mm-ddT hh:mm:ss</td>
<td>Pyyyy-mm-ddThh:mm:ss</td>
<td>P000--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 $N8601B. 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`
- `2012-xxTxx:xx:xx`
- `2012`

### Normalizing Duration Components

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

- 201209.5
- P2012-09-15T10.33
- 2012-09-15/P0003-03-03,333
Chapter 2
Formats Documented in Other Publications

## Chapter 3

**Dictionary of Formats for SAS and the CAS Server**

<table>
<thead>
<tr>
<th>Formats by Category</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>37</td>
</tr>
<tr>
<td>$BASE64X Format</td>
<td>37</td>
</tr>
<tr>
<td>$BINARY Format</td>
<td>38</td>
</tr>
<tr>
<td>$CHAR Format</td>
<td>39</td>
</tr>
<tr>
<td>$HEX Format</td>
<td>40</td>
</tr>
<tr>
<td>$OCTAL Format</td>
<td>41</td>
</tr>
<tr>
<td>$QUOTE Format</td>
<td>42</td>
</tr>
<tr>
<td>$REVERJ Format</td>
<td>43</td>
</tr>
<tr>
<td>$REVERS Format</td>
<td>44</td>
</tr>
<tr>
<td>$UPCASE Format</td>
<td>45</td>
</tr>
<tr>
<td>$ Format</td>
<td>46</td>
</tr>
<tr>
<td>BEST Format</td>
<td>46</td>
</tr>
<tr>
<td>BESTD Format</td>
<td>48</td>
</tr>
<tr>
<td>BESTDOTX Format</td>
<td>50</td>
</tr>
<tr>
<td>BINARY Format</td>
<td>51</td>
</tr>
<tr>
<td>COMMA Format</td>
<td>52</td>
</tr>
<tr>
<td>COMMAX Format</td>
<td>53</td>
</tr>
<tr>
<td>DATEAMPM Format</td>
<td>54</td>
</tr>
<tr>
<td>DATETIME Format</td>
<td>56</td>
</tr>
<tr>
<td>DAY Format</td>
<td>58</td>
</tr>
<tr>
<td>DDMMYY Format</td>
<td>58</td>
</tr>
<tr>
<td>DDMMYYx Format</td>
<td>60</td>
</tr>
<tr>
<td>DOLLAR Format</td>
<td>62</td>
</tr>
<tr>
<td>DOLLARX Format</td>
<td>63</td>
</tr>
<tr>
<td>DOWNAME Format</td>
<td>64</td>
</tr>
<tr>
<td>DTDATE Format</td>
<td>65</td>
</tr>
<tr>
<td>DTMONYY Format</td>
<td>66</td>
</tr>
<tr>
<td>DTWKPDX Format</td>
<td>68</td>
</tr>
<tr>
<td>DTYEAR Format</td>
<td>69</td>
</tr>
<tr>
<td>DTYYQC Format</td>
<td>70</td>
</tr>
<tr>
<td>E Format</td>
<td>71</td>
</tr>
<tr>
<td>FLOAT Format</td>
<td>72</td>
</tr>
<tr>
<td>FRACT Format</td>
<td>73</td>
</tr>
<tr>
<td>HEX Format</td>
<td>74</td>
</tr>
<tr>
<td>HHMM Format</td>
<td>75</td>
</tr>
<tr>
<td>HOUR Format</td>
<td>77</td>
</tr>
<tr>
<td>IEEE Format</td>
<td>78</td>
</tr>
<tr>
<td>JULIAN Format</td>
<td>80</td>
</tr>
<tr>
<td>MDYAMPM Format</td>
<td>81</td>
</tr>
<tr>
<td>MMDDYY Format</td>
<td>82</td>
</tr>
</tbody>
</table>
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 Viya 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 *SAS Viya Utility Procedures Guide*.

The following table provides brief descriptions of the SAS formats that are supported in traditional SAS and CAS. 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>SBASE64X Format (p. 37)</td>
<td>Converts character data into ASCII text by using Base 64 encoding.</td>
</tr>
<tr>
<td></td>
<td>$BINARY Format (p. 38)</td>
<td>Converts character data to binary representation.</td>
</tr>
<tr>
<td></td>
<td>$CHAR Format (p. 39)</td>
<td>Writes standard character data.</td>
</tr>
<tr>
<td></td>
<td>SHEX Format (p. 40)</td>
<td>Converts character data to hexadecimal representation.</td>
</tr>
<tr>
<td></td>
<td>SOCTAL Format (p. 41)</td>
<td>Converts character data to octal representation.</td>
</tr>
<tr>
<td></td>
<td>SQUOTE Format (p. 42)</td>
<td>Writes data values that are enclosed in double quotation marks.</td>
</tr>
<tr>
<td></td>
<td>$REVERJ Format (p. 43)</td>
<td>Writes character data in reverse order and preserves blanks.</td>
</tr>
<tr>
<td></td>
<td>$REVERS Format (p. 44)</td>
<td>Writes character data in reverse order and left aligns</td>
</tr>
<tr>
<td></td>
<td>$UPCASE Format (p. 45)</td>
<td>Converts character data to uppercase.</td>
</tr>
<tr>
<td></td>
<td>$ Format (p. 46)</td>
<td>Writes standard character data.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>DATEAMPM Format (p. 54)</td>
<td>Writes datetime values in the form ddmmmyy:hh:mm:ss.ss with AM or PM.</td>
</tr>
<tr>
<td></td>
<td>DATETIME Format (p. 56)</td>
<td>Writes datetime values in the form ddmmmyy:hh:mm:ss.ss.</td>
</tr>
<tr>
<td></td>
<td>DAY Format (p. 58)</td>
<td>Writes date values as the day of the month.</td>
</tr>
<tr>
<td></td>
<td>DDMMYY Format (p. 58)</td>
<td>Writes date values in the form ddmm&lt;yy&gt;yy or dd/mm&lt;yy&gt;&lt;yy&gt;, where a forward slash is the separator and the year appears as either 2 or 4 digits.</td>
</tr>
<tr>
<td></td>
<td>DDMMYYx Format (p. 60)</td>
<td>Writes date values in the form ddmm&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>
</tr>
<tr>
<td></td>
<td>DOWNAME Format (p. 64)</td>
<td>Writes date values as the name of the day of the week.</td>
</tr>
<tr>
<td></td>
<td>DTDATE Format (p. 65)</td>
<td>Expects a datetime value as input and writes date values in the form ddmmmyy or ddmmmyyyy.</td>
</tr>
<tr>
<td></td>
<td>DTMONYY Format (p. 66)</td>
<td>Writes the date part of a datetime value as the month and year in the form mmmyy or mmmyyyy.</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 Format (p. 68)</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>
<td></td>
</tr>
<tr>
<td>DTYEAR Format (p. 69)</td>
<td>Writes the date part of a datetime value as the year in the form yy or yyyy.</td>
<td></td>
</tr>
<tr>
<td>DTYYQC Format (p. 70)</td>
<td>Writes the date part of a datetime value as the year and the quarter and separates them with a colon (:).</td>
<td></td>
</tr>
<tr>
<td>HHMM Format (p. 75)</td>
<td>Writes time values as hours and minutes in the form hh:mm.</td>
<td></td>
</tr>
<tr>
<td>HOUR Format (p. 77)</td>
<td>Writes time values as hours and decimal fractions of hours.</td>
<td></td>
</tr>
<tr>
<td>JULIAN Format (p. 80)</td>
<td>Writes date values as Julian dates in the form yyyddd or yyyyddd.</td>
<td></td>
</tr>
<tr>
<td>MDYAMPM Format (p. 81)</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>MMDDYY Format (p. 82)</td>
<td>Writes date values in the form mm/dd/yy&lt;yy&gt; yy or mm/dd/yy, where a forward slash is the separator and the year appears as either 2 or 4 digits.</td>
<td></td>
</tr>
<tr>
<td>MMDDYYx Format (p. 84)</td>
<td>Writes date values in the form mm/dd&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>MMSS Format (p. 86)</td>
<td>Writes time values as the number of minutes and seconds since midnight.</td>
<td></td>
</tr>
<tr>
<td>MMYY Format (p. 88)</td>
<td>Writes date values in the format mm/M&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>MMYYx Format (p. 89)</td>
<td>Writes date values in the format 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>MONNAME Format (p. 91)</td>
<td>Writes date values as the name of the month.</td>
<td></td>
</tr>
<tr>
<td>MONTH Format (p. 92)</td>
<td>Writes date values as the month of the year.</td>
<td></td>
</tr>
<tr>
<td>MONYY Format (p. 92)</td>
<td>Writes date values as the month and the year in the form mmmmyy or mmmmyyyy.</td>
<td></td>
</tr>
<tr>
<td>QTR Format (p. 96)</td>
<td>Writes date values as the quarter of the year.</td>
<td></td>
</tr>
<tr>
<td>QTRR Format (p. 97)</td>
<td>Writes date values as the quarter of the year in Roman numerals.</td>
<td></td>
</tr>
<tr>
<td>TIME Format (p. 98)</td>
<td>Writes time values as hours, minutes, and seconds in the form hh:mm:ss.ss.</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>TIMEAMPM Format</td>
<td>(p. 100)</td>
<td>Writes time and datetime values as hours, minutes, and seconds in the form hh:mm:ss.ss with AM or PM.</td>
</tr>
<tr>
<td>TOD Format</td>
<td>(p. 102)</td>
<td>Writes SAS time values and the time portion of SAS datetime values in the form hh:mm:ss.ss.</td>
</tr>
<tr>
<td>WEEKDATE Format</td>
<td>(p. 106)</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>
</tr>
<tr>
<td>WEEKDATX Format</td>
<td>(p. 107)</td>
<td>Writes date values as the day of the week and date in the form day-of-week, dd month-name yy (or yyyy).</td>
</tr>
<tr>
<td>WEEKDAY Format</td>
<td>(p. 109)</td>
<td>Writes date values as the day of the week.</td>
</tr>
<tr>
<td>WEEKU Format</td>
<td>(p. 110)</td>
<td>Writes a week number in decimal format by using the U algorithm.</td>
</tr>
<tr>
<td>WEEKV Format</td>
<td>(p. 111)</td>
<td>Writes a week number in decimal format by using the V algorithm.</td>
</tr>
<tr>
<td>WEEKW Format</td>
<td>(p. 113)</td>
<td>Writes a week number in decimal format by using the W algorithm.</td>
</tr>
<tr>
<td>YEAR Format</td>
<td>(p. 115)</td>
<td>Writes date values as the year.</td>
</tr>
<tr>
<td>YYMM Format</td>
<td>(p. 116)</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>
</tr>
<tr>
<td>YYMMDD Format</td>
<td>(p. 117)</td>
<td>Writes date values in the form yymmdd or &lt;yy&gt;yy-mm-dd, where a hyphen is the separator and the year appears as either 2 or 4 digits.</td>
</tr>
<tr>
<td>YYMMDDx Format</td>
<td>(p. 119)</td>
<td>Writes date values in the form yymmdd 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>
</tr>
<tr>
<td>YYMOM Format</td>
<td>(p. 121)</td>
<td>Writes date values in the form yyyyymm or yyyymm.</td>
</tr>
<tr>
<td>YYQ Format</td>
<td>(p. 122)</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>
</tr>
<tr>
<td>YYQx Format</td>
<td>(p. 123)</td>
<td>Writes date values in the form &lt;yy&gt;yyq or &lt;yy&gt;yy-q, where the x in the format name is a character that represents the special character that separates the year and the quarter or the year, which can be a hyphen (-), period (.), blank character, slash (/), colon (:), or no separator; the year can be either 2 or 4 digits.</td>
</tr>
<tr>
<td>YYQR Format</td>
<td>(p. 125)</td>
<td>Writes date values in the form &lt;yy&gt;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.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>YYQRx Format (p. 126)</td>
<td>Writes date values in the form &lt;yy&gt;yyqr or &lt;yy&gt;yy-qr, where the x in the format name is a character that represents the special character 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.</td>
<td></td>
</tr>
<tr>
<td>Numeric</td>
<td>BEST Format (p. 46)</td>
<td>SAS chooses the best notation.</td>
</tr>
<tr>
<td></td>
<td>BESTD Format (p. 48)</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>BESTDOTX Format (p. 50)</td>
<td>Specifies that SAS choose the best notation and use a dot as a decimal separator.</td>
</tr>
<tr>
<td></td>
<td>BINARY Format (p. 51)</td>
<td>Converts numeric values to binary representation.</td>
</tr>
<tr>
<td></td>
<td>COMMA Format (p. 52)</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 Format (p. 53)</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>DOLLAR Format (p. 62)</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 Format (p. 63)</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 Format (p. 71)</td>
<td>Writes numeric values in scientific notation.</td>
</tr>
<tr>
<td></td>
<td>FLOAT Format (p. 72)</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 Format (p. 73)</td>
<td>Converts numeric values to fractions.</td>
</tr>
<tr>
<td></td>
<td>HEX Format (p. 74)</td>
<td>Converts real binary (floating-point) values to hexadecimal representation.</td>
</tr>
<tr>
<td></td>
<td>IEEE Format (p. 78)</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 Format (p. 94)</td>
<td>Writes negative numeric values in parentheses.</td>
</tr>
<tr>
<td></td>
<td>OCTAL Format (p. 95)</td>
<td>Converts numeric values to octal representation.</td>
</tr>
<tr>
<td></td>
<td>ROMAN Format (p. 97)</td>
<td>Writes numeric values as roman numerals.</td>
</tr>
<tr>
<td></td>
<td>VAXRB Format (p. 104)</td>
<td>Writes real binary (floating-point) data in VMS format.</td>
</tr>
<tr>
<td></td>
<td>w Format (p. 105)</td>
<td>Writes standard numeric data one digit per byte.</td>
</tr>
</tbody>
</table>
$BASE64X Format

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

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

### Syntax

$BASE64X_w$

### Syntax Description

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

  You can use the following formula to determine the width:

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

  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 \(\frac{48+2}{3} \times 4 = 64\).

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

**Default** 1  
**Range** 1–32767

### Details

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

Here are some uses of Base 64 encoding:

- embed binary data in an XML file
- encode passwords
- encode URLs
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>TXIQYXNzd29yZA=</td>
</tr>
<tr>
<td>&quot;www.mydomain.com/myhiddenURL&quot;</td>
<td>d3d3Lm15ZG9tYWluLmNvbib9teWhpZGRlblVSTA=</td>
</tr>
</tbody>
</table>

See Also


Informats:

- “$BASE64X Informat” on page 261

$BINARY Format

Converts character data to binary representation.

**Category:** Character

**Alignment:** Left

**Syntax**

```sas
$BINARY w.
```

**Syntax Description**

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

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

  - **Range** 1–32767

**Comparisons**

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

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

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

$CHAR 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.

Example

```
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>

---

Example

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

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

$CHAR 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.

Example

```
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>
**$HEX Format**

Converts character data to hexadecimal representation.

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

---

**Syntax**

$HEX_{w.}$

**Syntax Description**

$w$

specifies the width of the output field.

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

- **Range**  
  1–32,767

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

  If $w$ is greater than twice the length of the variable that you want to represent, $HEX_{w.}$ pads it with blanks.

---

**Details**

The $HEX_{w.}$ format produces hexadecimal representations of ASCII codes for characters. Each byte requires two columns. Therefore, you need twice as many columns to output a value with the $HEX_{w.}$ format.

---

**Comparisons**

The 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>ASCII</td>
<td></td>
</tr>
<tr>
<td>AB</td>
<td>4142</td>
</tr>
</tbody>
</table>
$OCTAL$ Format

Converts character data to octal representation.

**Category:** Character

**Alignment:** Left

---

**Syntax**

$OCTALw$.

---

**Syntax Description**

$w$

specifies the width of the output field.

**Default**

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

**Range**

1–32767

**Tip**

Because each character value generates three octal characters, increase the value of $w$ by three times the length of the character value.

---

**Comparisons**

The $OCTALw$ format converts character values to the octal representation of their character codes. The OCTALw format converts numeric values to octal representation.

---

**Example**

The following example shows ASCII output when you use the $OCTALw$ 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
142141156153040
```

```
$QUOTE Format

Writes data values that are enclosed in double quotation marks.

Category:  Character
Alignment:  Left

Syntax

$QUOTEw.

Syntax Description

$QUOTEw.

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

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

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

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

$REVERJ Format

Writes character data in reverse order and preserves blanks.

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

**Syntax**

```sas
$REVERJw.
```

**Syntax Description**

`w`

specifies the width of the output field.

**Default**  
1, if `w` is not specified

**Range**  
1–32767
Comparisons

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

Example

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

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

* The character # represents a blank space.

$REVERS Format

Writes character data in reverse order and left aligns

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

Syntax

$REVERS\ w.

**Syntax Description**

- \( w \)
  - specifies the width of the output field.
  - **Default:** 1 if \( w \) is not specified
  - **Range:** 1–32767

Comparisons

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

Example

```sas
put @1 name $revers7.;
```
### $UPCASE Format

Converts character data to uppercase.

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

#### Syntax

$UPCASEw$.  

#### Syntax Description

<table>
<thead>
<tr>
<th>$w$</th>
<th>Specifies the width of the output field.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>8, if the length of the variable is undefined. Otherwise, the default is the length of the variable</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>1–32767</td>
</tr>
</tbody>
</table>

#### Details

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

#### Example

```
put 01 name $upcase9.;
```

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

```sas
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.

BEST Format

SAS chooses the best notation.

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 Viya System Options: Reference.

Syntax

```
BESTw.
```

Syntax Description

```
w
```

Specifies the width of the output field.

Default  12

Range  1–32

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

Details

When a format is not specified for writing a numeric value, SAS uses the 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 Viya 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 D\_w.p format writes numbers with the desired precision and more alignment than the BEST\_w format.

The BESTD\_w.p format is a combination of the BEST\_w format and the D\_w.p format in that it formats all numeric data, and it does a better job of aligning decimals than the BEST\_w format.

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

**Example**

The following statements produce these results:

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

**See Also**

**Formats:**

- “BESTD Format” on page 48

---

**BESTD 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 STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV= System Option” in SAS Viya System Options: Reference.

**Syntax**

BESTDw.p

**Syntax Description**

w

specifies the width of the output field.
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
```fortran
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>123.450000</td>
</tr>
<tr>
<td>1.2345</td>
<td>1.23450000</td>
</tr>
</tbody>
</table>
### BESTDOTX Format

Specifies that SAS choose the best notation and use a dot as a decimal separator.

**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 Viya System Options: Reference.

**Syntax**

`BESTDOTX.w`

**Arguments**

- **w** specifies the width of the output field.
  
  **Default** 12  
  **Range** 1–32  
  **Tip** If you print numbers between 0 and .01 exclusively, then use a field width of at least 7 to avoid excessive rounding. If you print numbers between 0 and -.01 exclusively, use a field width of at least 8.

### Details

If the NLDECSEPARATOR system option is disabled, the BESTw. and BESTDOTXw. formats process data the same way. If the NLDECSEPARATOR system option is enabled, then the results from the BESTw. and BESTDOTXw. formats are different. See the following table to understand the differences:

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>.12345</td>
<td>0.1234500</td>
</tr>
<tr>
<td>1.23456789</td>
<td>1.2345679</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “BEST Format” on page 46  
- “D Format” on page 140
<table>
<thead>
<tr>
<th>LOCALE option</th>
<th>Default decimal separator character for the locale</th>
<th>NLDECSEPARATOR option</th>
<th>Separator character used by BESTw.</th>
<th>Separator character used by BESTDOTXw.</th>
</tr>
</thead>
<tbody>
<tr>
<td>en_US</td>
<td>Dot</td>
<td>Disabled (default)</td>
<td>Dot</td>
<td>Dot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enabled</td>
<td>Dot</td>
<td>Dot</td>
</tr>
<tr>
<td>fr_FR</td>
<td>Comma</td>
<td>Disabled (default)</td>
<td>Dot</td>
<td>Dot</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enabled</td>
<td>Comma</td>
<td>Dot</td>
</tr>
</tbody>
</table>

For more information about the NLDECSEPARATOR system option, see *SAS Viya National Language Support (NLS): Reference Guide*.

**Comparisons**

The BESTDOTXw format writes as many significant digits as possible in the output field. Integers are printed with a decimal. 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.

**See Also**

**Formats:**

- “BEST Format” on page 46

---

**BINARY Format**

Converts numeric values to binary representation.

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

**Syntax**

`BINARY w`.

**Syntax Description**

`w`

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>1–64</td>
</tr>
</tbody>
</table>

**Comparisons**

`BINARYw` converts numeric values to binary representation. The `SBINARYw` format converts character values to binary representation.
**Example**

\[
\text{put } @1 x \text{ binary8.};
\]

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

**COMMA 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 STDIEEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV= System Option” in SAS Viya System Options: Reference.

**Syntax**

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

**Syntax Description**

\( w \)

specifies the width of the output field.

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

\( d \)

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

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

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

Comparisons

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

• The COMMA\textit{w.d} format is similar to the DOLLAR\textit{w.d} format except that the COMMA\textit{w.d} format does not print a leading dollar sign.

Example

\begin{verbatim}
put @10 sales comma10.2;
\end{verbatim}

<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:

• “COMMAX Format” on page 53
• “DOLLAR Format” on page 62

COMMAX Format

Writes numeric values with 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 Viya System Options: Reference. |

Syntax

COMMAX\textit{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 \( w \)

**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**

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

<table>
<thead>
<tr>
<th>Value of sales</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>---+-------2</td>
</tr>
<tr>
<td>23451.23</td>
<td>23.451,23</td>
</tr>
<tr>
<td>123451.234</td>
<td>123.451,23</td>
</tr>
</tbody>
</table>

---

**DATEAMPM Format**

Writes datetime values in the form \( ddmnmyy:hh:mm:ss.ss \) 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 Viya System Options: Reference.

**Syntax**

\[ DATEAMPM w.d \]
**Syntax Description**

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

- **d**
  - Specifies the number of digits to the right of the decimal point in the seconds value.
  - This argument is optional.
  - Range: 0–39
  - Requirement: 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
```

- **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></td>
<td>- - - - -1- - - - -2- - - - +</td>
</tr>
</tbody>
</table>


### SAS Statement

<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>20APR11</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:**

- “DATETIME Format” on page 56

### DATETIME 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 Viya System Options: Reference.

### Syntax

**DATETIME**<w,d>

**Syntax Description**

- `<w>` specifies the width of the output field.
  - **Default:** 16
  - **Range:** 7–40
  - **Tip:** SAS requires a minimum `w` value of 16 to write a SAS datetime value with the date, hour, and seconds. Add an additional two places to `w` and a value to `d` to return values with optional decimal fractions of seconds.

- `<d>` specifies the number of digits to the right of the decimal point in the seconds value.
  - This argument is optional.
Details

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

- $dd$ is an integer that represents the day of the month.
- $mmm$ is the first three letters of the month name.
- $yy$ is a two-digit integer that represents the year.
- $hh$ is an integer that represents the hour in 24–hour clock time.
- $mm$ is an integer that represents the minutes.
- $ss.ss$ is the number of seconds to two decimal places.

Example

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>
<tr>
<td>put event datetime20.1;</td>
<td>10NOV2012:03:49:19.0</td>
</tr>
<tr>
<td>put event datetime21.2;</td>
<td>10NOV2012:03:49:19.00</td>
</tr>
</tbody>
</table>

See Also

Formats:
DAY Format

Writes date values as the day of the month.

**Syntax**

```
DAY <w>.
```

**Syntax Description**

- `<w>` specifies the width of the output field.  
  
  **Default:** 2  
  **Range:** 2–32

**Example**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date day2.;</td>
<td>14</td>
</tr>
</tbody>
</table>

DDMMYY Format

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

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

DDMMYYw.

Syntax Description

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

Details

The DDMMYYw. format writes SAS date values in the form ddmmyyy or dd/mm/<yy>yy:

- **dd** is an integer that represents the day of the month.
- **/** is the separator.
- **mm** is an integer that represents the month.
- **<yy>yy** is a two-digit or four-digit integer that represents the year.

Example

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date ddmmyy5.;</td>
<td>24/12</td>
</tr>
<tr>
<td>put date ddmmyy6.;</td>
<td>241212</td>
</tr>
<tr>
<td>put date ddmmyy7.;</td>
<td>241212</td>
</tr>
<tr>
<td>put date ddmmyy8.;</td>
<td>24/12/12</td>
</tr>
<tr>
<td>put date ddmmyy10.;</td>
<td>24/12/2012</td>
</tr>
</tbody>
</table>
**DDMMYYx Format**

Writes date values in the form `ddm<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.

**Default** 8
Range | 2–10  
---|---  
Interactions | When \( w \) has a value of from 2 to 5, the date appears with as much of the day and the month as possible. When \( w \) is 7, the date appears as a two-digit year without separators.  
| When \( x \) has a value of \( N \), the width range changes to 2–8.

**Details**

The DDMMYYxw. format writes SAS date values in the form \( ddmm<yy>yy \) or \( ddxmxx<yy>yy \):

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

**Example**

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

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

**See Also**

**Formats:**

- “DATE Format” on page 141
- “DDMMYY Format” on page 58
- “MMDDYYx Format” on page 84
- “YYMMDDx Format” on page 119

**Functions:**

- “DAY Function” in *SAS Viya Functions and CALL Routines: Reference*
DOLLAR 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 Viya System Options: Reference.

### Syntax

DOLLAR\textit{w.d}

### Syntax Description

\textit{w}

specifies the width of the output field.

- **Default:** 6
- **Range:** 2–32

\textit{d}

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

- **Range:** 0–31

### Details

The DOLLAR\textit{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\textit{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 SAS Viya Utility Procedures Guide.
Comparisons

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

Example

```plaintext
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 Format” on page 52
- “DOLLARX Format” on page 63

DOLLARX 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 STDIEEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV= System Option” in SAS Viya System Options: Reference.

Syntax

DOLLARXw.d

**Syntax Description**

w

specifies the width of the output field.

**Default** 6  
**Range** 2–32
\(d\)

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

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

**Details**

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

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

**Comparisons**

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

**Example**

```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:**

- “COMMAX Format” on page 53
- “DOLLAR Format” on page 62

**DOWNNAME Format**

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

**DOWNAME**_w_.

**Syntax Description**

_w_ specifies the width of the output field.

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

**Details**

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

**Example**

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:**

- “WEEKDAY Format” on page 109
Syntax Description

\( w \)

specifies the width of the output field.

Default 7

Range 5–9

Tip Use a width of 9 to print a four-digit year.

Details

The DTDATE\( w \) format writes SAS date values in the form \( ddmmmyy \) or \( ddmmmyyyy \), where

\( dd \)

is an integer that represents the day of the month.

\( mmm \)

are the first three letters of the month name.

\( yy \) or \( yyyy \)

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

Comparisons

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

Example

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

<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:

• “DATE Format” on page 141

DTMONYY Format

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

Category: Date and Time
**Syntax**

\texttt{DTMONYYw.}

**Syntax Description**

\( w \)

specifies the width of the output field.

- **Default**: 5
- **Range**: 5–7

**Details**

The \texttt{DTMONYYw.} format writes SAS datetime values in the form \textit{mmm}yy or \textit{mmm}yyyy, where

- \textit{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 \texttt{DTMONYYw.} format and the \texttt{MONYYw.} format are similar in that they both write date values. The difference is that \texttt{DTMONYYw.} expects a datetime value as input, and \texttt{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>\texttt{put date dtmonyy.;}</td>
<td>OCT12</td>
</tr>
<tr>
<td>\texttt{put date dtmonyy5.;}</td>
<td>OCT12</td>
</tr>
<tr>
<td>\texttt{put date dtmonyy6.;}</td>
<td>OCT12</td>
</tr>
<tr>
<td>\texttt{put date dtmonyy7.;}</td>
<td>OCT2012</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “DATETIME Format” on page 56
DTWKDATX 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

\texttt{DTWKDATXw.}

### Syntax Description

- \texttt{w} specifies the width of the output field.
  - **Default:** 29
  - **Range:** 3–37

### Details

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

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

### Comparisons

The \texttt{DTWKDATXw.} format is similar to the \texttt{WEEKDATXw.} format in that they both write date values. The difference is that \texttt{DTWKDATXw.} expects a datetime value as input, and \texttt{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, 20012, at 06:08:52 a.m.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- - - - - 1 - - - - - 2 - - - - -</td>
</tr>
</tbody>
</table>
### DTYEAR Format

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

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

### Syntax

`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:**
- “DATETIME Format” on page 56
- “YEAR Format” on page 115

---

**DTYYQC 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.

**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..
SAS Statement | Result
---|---
| | ----+----1
| put date dtyyqc.; | 12:4
| put date dtyq4.; | 12:4
| put date dtyyqc5.; | 12:4
| put date dtyqc6.; | 2012:4

**See Also**

**Formats:**
- “DATETIME Format” on page 56

---

**E Format**

Writes numeric values in scientific notation.

**Category:** Numeric

**Alignment:** Right

**Syntax**

\texttt{E w.}

**Syntax Description**

\texttt{w}

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

- **Default:** 12
- **Range:** 7–32

**Details**

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

**Example**

\texttt{put @1 x e10.;}
FLOAT Format

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

**Category:** Numeric

**Alignment:** Left

**Syntax**

```plaintext
FLOATw.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 LINUX, 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.

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

---
Comparisons

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

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

Example

```plaintext
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.

FRACT 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

```
put x fract8.;
```

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.6666666667</td>
<td>2/3</td>
</tr>
<tr>
<td>0.2784</td>
<td>174/625</td>
</tr>
</tbody>
</table>

HEX Format

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

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

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

The HEXw. format converts a real (floating-point) binary number to its hexadecimal representation. When you specify a width value of 1 through 15, the real binary number is truncated to a fixed-point integer before being converted to a hexadecimal number. When you specify 16 for the width, SAS writes the floating-point value of the number, but does not truncate it.
In any operating environment, the least significant byte written by HEXw. is the rightmost 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 hex8.;
```

<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>

**HHMM 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 STDEEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV= System Option” in SAS Viya 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 HHMM\(w.d\) format writes SAS time values in the form \( hh:mm \):

\( hh \)
is an integer.

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

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

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

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

Comparisons

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

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

Example

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

<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:
- “HOUR Format” on page 77
- “MMSS Format” on page 86
- “TIME Format” on page 98
- “TOD Format” on page 102

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

Informats:
- “TIME Informat” on page 354

HOUR Format

Writes time values as hours and decimal fractions of hours.

**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 Viya System Options: Reference.

**Syntax**

\[
\text{HOUR}^{w,d}
\]

**Syntax Description**

\(w\)

specifies the width of the output field.

- **Default:** 2
- **Range:** 2–20

\(d\)

specifies the number of digits to the right of the decimal point in the hour value. Therefore, SAS prints decimal fractions of the hour. This argument is optional.
Range: 0–19

Requirement: must be less than $w$

Details
SAS rounds hours based on the value of minutes in the SAS time value.
The HOUR format uses asterisks to format values that are outside the time range 0–24
hours, such as datetime values.

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put time hour4.1;</td>
<td>11.5</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “HHMM Format” on page 75
- “MMSS Format” on page 86
- “TIME Format” on page 98
- “TOD Format” on page 102

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

Informats:
- “TIME Informat” on page 354

IEEE Format
Generates an IEEE floating-point value by multiplying a number by 10 raised to the dth power.

Category: Numeric
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 IEEEw.d is the floating-point representation that is used. In addition, you can use the IEEEw.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 IEEEw.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.;
```
Value of \( x \) | Result
--- | ---
1 | 3F800000
   | 3F00000000

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

**JULIAN Format**

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

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

**Syntax**

\[ \text{JULIAN} \_w. \]

**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>
</tbody>
</table>

---

Chapter 3 • Dictionary of Formats for SAS and the CAS Server
SAS Statement | Result
--- | ---
puts date julian7.; | 2012122

**See Also**

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

**Informats:**
- “JULIAN Informat” on page 315

---

**MDYAMPm Format**

Writes datetime values in the form `mm/dd<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 STDIEEE, the output that is written using this format might differ slightly from previous releases. For more information, see “DECIMALCONV= System Option” in *SAS Viya System Options: Reference*.

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

**Syntax**

**Syntax Description**

\[ \text{MDYAMPm}_w \]

w specifies the width of the output field.

- **Default** 19  
- **Range** 8−40

**Details**

The MDYAMPm\_d format writes SAS datetime values in the following form:

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

- mm is an integer between 1 and 12 that represents the month.
- dd is an integer between 1 and 31 that represents the day of the month.
**yy or yyyy**
specifies a two-digit or four-digit integer that represents the year.

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

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

**AM | PM**
specifies either the time period 00:01–12:00 noon (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 MDYAMPMPm format writes datetime values with separators in the form `mm/dd/yyyy hh:mm AM | PM`, and requires a space between the date and the time.

The DATETIMEw.d format writes datetime values with separators in the form `ddmmmyyy yyyy: 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:**
- “DATETIME Format” on page 56

**Informats:**
- “MDYAMPMP Informat” on page 316

**MMDDYY Format**
Writes date values in the form `mmd<yy>yy` or `mm/dd/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.

Default  8

Range     2–10

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

Details

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

\[
\text{mmdd}<\text{yy}\text{yy}
\]

\[
\text{mm/dd}<\text{yy}\text{yy}:
\]

where

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

\[
/
\]
is the separator.

\[
\text{dd}
\]
is an integer that represents the day of the month.

\[
<\text{yy}\text{yy}
\]
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>
</tbody>
</table>
MMDDYYx Format

Writes date values in the form \textit{mmdd<\textless yy>yy} or \textit{mm-dd<\textless yy>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}

\textbf{Syntax}

\texttt{MMDDYY\_xw}. 

\textbf{See Also}

\textbf{Formats:}

\begin{itemize}
  \item “DATE Format” on page 141
  \item “DDMMYY Format” on page 58
  \item “MMDDYYx Format” on page 84
  \item “YYMMDD Format” on page 117
\end{itemize}

\textbf{Functions:}

\begin{itemize}
  \item “DAY Function” in \textit{SAS Viya Functions and CALL Routines: Reference}
  \item “MDY Function” in \textit{SAS Viya Functions and CALL Routines: Reference}
  \item “MONTH Function” in \textit{SAS Viya Functions and CALL Routines: Reference}
  \item “YEAR Function” in \textit{SAS Viya Functions and CALL Routines: Reference}
\end{itemize}

\textbf{Informats:}

\begin{itemize}
  \item “DATE Informat” on page 293
  \item “DDMMYY Informat” on page 296
  \item “YYMMDD Informat” on page 362
\end{itemize}
**Syntax Description**

\( x \)

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

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

\( w \)

specifies the width of the output field.

**Default** 8

**Range** 2–10

**Interactions**

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

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

**Details**

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

\[
\text{mmdd<yy>yy} \\
\text{mmxDD<yy>yy}
\]

where

**mm**

is an integer that represents the month.

**x**

is a specified separator.

**dd**

is an integer that represents the day of the month.

\(<yy>yy\)

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

**Example**

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

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

<table>
<thead>
<tr>
<th>Category</th>
<th>Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment</td>
<td>Right</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 Viya System Options: Reference.</td>
</tr>
</tbody>
</table>

### Syntax

```sas
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>

**See Also**

**Formats:**
- “HHMM Format” on page 75
- “TIME Format” on page 98

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

**Informats:**
- “TIME Informat” on page 354
**MMYY Format**

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

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

**Syntax**

`MMYYw`.

**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 `MMYYw` format writes SAS date values in the form `mmM<yy>yy`, where

- `mm` is an integer that represents the month.
- `M` is the character separator.
- `<yy>yy` is a two-digit or four-digit integer that represents the year.

**Example**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>put date mmyy5.;</code></td>
<td>10M12</td>
</tr>
<tr>
<td><code>put date mmyy6.;</code></td>
<td>10M12</td>
</tr>
<tr>
<td><code>put date mmyy.;</code></td>
<td>10M2012</td>
</tr>
</tbody>
</table>
## MMYYx Format

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

### Category:
Date and Time

### Alignment:
Right

## Syntax

`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.

### Default

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

### Range

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Range</strong></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 MMYY.xw. format writes SAS date values in one of the following forms:

\[
\text{\textit{mm}<yy>yy} \\
\text{\textit{mmx<yy>yy}}
\]

where

\( \text{\textit{mm}} \)

is an integer that represents the month.

\( x \)

is a specified separator.

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

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 date mmyyc5.;</td>
<td>05:12</td>
</tr>
<tr>
<td>put date mmyyd.;</td>
<td>05-2012</td>
</tr>
<tr>
<td>put date mmyyn4.;</td>
<td>0512</td>
</tr>
<tr>
<td>put date mmyyp8.;</td>
<td>05.2012</td>
</tr>
<tr>
<td>put date mmyys10.;</td>
<td>05/2012</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “MMYY Format” on page 88
- “YYMMx Format” on page 226
MONNAME 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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date monname1.;</td>
<td>M</td>
</tr>
<tr>
<td>put date monname3.;</td>
<td>Mar</td>
</tr>
<tr>
<td>put date monname5.;</td>
<td>March</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “MONTH Format” on page 92
MONTH 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.

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

### See Also

**Formats:**  
- “MONNAME Format” on page 91

---

MONYY Format

Writes date values as the month and the year in the form mmmyy or mmmyyyy.
Category: Date and Time
Alignment: Right

Syntax

MONYYw.

Syntax Description

w

specifies the width of the output field.

Default  5
Range   5–7

Details

The MONYYw. format writes SAS date values in the form mmmyy or mmmyyyy, 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.

<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>

See Also

Formats:
- “DDMMYY Format” on page 58
- “DTMONYY Format” on page 66
- “MMDDYY Format” on page 82
• “YYMMDD Format” on page 117

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

Informat:
• “MONYY Informat” on page 319

---

**NEGPAREN 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 Viya System Options: Reference*.

---

**Syntax**

NEGPARENw.d

**Syntax Description**

- **w**
  - specifies the width of the output field.
  - Default: 6
  - Range: 1–32

- **d**
  - specifies the number of digits to the right of the decimal point in the numeric value. This argument is optional.
  - Default: 0
  - Range: 0–31

**Details**

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

The 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>

OCTAL Format

Converts numeric values to octal representation.

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

Syntax

`OCTAL$w.$`

Syntax Description

$w$

specifies the width of the output field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1–24</td>
</tr>
</tbody>
</table>

Details

If necessary, the OCTAL$w.$ format converts numeric values to integers before displaying them in octal representation.

Comparisons

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

```
put x octal6.;
```

<table>
<thead>
<tr>
<th>Value of (x)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(3592)</td>
<td>(007010)</td>
</tr>
</tbody>
</table>

**QTR Format**

Writes date values as the quarter of the year.

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

**Syntax**

\(QTR^w\).

**Syntax Description**

\(^w\)

specifies the width of the output field.

- **Default:** 1
- **Range:** 1–32

**Example**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{put date qtr.};)</td>
<td>1</td>
</tr>
</tbody>
</table>

**See Also**

- “QTRR Format” on page 97
**QTRR Format**

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

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

---

**Syntax**

```
QTRR w.
```

**Syntax Description**

`w` specifies the width of the output field.

**Default** 3  
**Range** 3–32

---

**Example**

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

<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:**  
- “QTR Format” on page 96

---

**ROMAN Format**

Writes numeric values as roman numerals.

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

---

**Syntax**

```
ROMAN w.
```
Syntax Description

\[ w \]

specifies the width of the output field.

Default: 6

Range: 2–32

Details

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

Example

\[
\text{put @5 year roman10.;}
\]

<table>
<thead>
<tr>
<th>Value of year</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012</td>
<td>MMXII</td>
</tr>
</tbody>
</table>

TIME 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 Viya System Options: Reference.

Syntax

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

Syntax Description

\[ w \]

specifies the width of the output field.

Default: 8

Range: 2–20

Tip: Make \( w \) large enough to produce the desired results. To obtain a complete time value with three decimal places, you must allow at least 12 spaces: eight spaces to the left of the decimal point, one space for the decimal point, and three spaces for the decimal fraction of seconds.
specifies the number of digits to the right of the decimal point in the seconds value. This argument is optional.

<table>
<thead>
<tr>
<th>Default</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>0–19</td>
</tr>
<tr>
<td>Requirement</td>
<td>must be less than ( w )</td>
</tr>
</tbody>
</table>

**Details**

The `TIME_{w,d}` format writes SAS time values in the form \( hh:mm:ss.ss \):

- \( hh \) is an integer.
  
  *Note:* If \( hh \) is a single digit, `TIME_{w,d}` places a leading blank before the digit. For example, the `TIME_{w,d}` format writes 9:00 instead of 09:00.

- \( mm \) is an integer between 00 and 59 that represents minutes.

- \( ss.ss \) is the number of seconds between 00 and 59, with the fraction of a second following the decimal point.

**Comparisons**

The `TIME_{w,d}` format is similar to the `HHMM_{w,d}` format except that `TIME_{w,d}` includes seconds.

The `TIME_{w,d}` format writes a leading blank for a single-hour digit. The `TOD_{w,d}` format writes a leading zero for a single-hour digit.

**Examples**

**Example 1**

This example uses the input value of 59083, which is the SAS time value that corresponds to 4:24:43 p.m.

<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>16:24:43</td>
</tr>
</tbody>
</table>
SAS Statement | Result
---|---
put begin time.; | 8:54:43

See Also

Formats:
- “HHMM Format” on page 75
- “HOUR Format” on page 77
- “MMSS Format” on page 86
- “TOD Format” on page 102

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

Informats:
- “TIME Informat” on page 354

**TIMEAMPM 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 Viya System Options: Reference*.

**Syntax**

```
TIMEAMPMw.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 TIMEAMPMw.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 TIMEAMPMw.d format is similar to the TIMEmw.d format except, that TIMEAMPMw.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:
- “TIME Format” on page 98

TOD Format
Writes SAS time values and the time portion of SAS datetime values in the form \( hh:mm:ss.ss \).

Category: Date and Time
Alignment: Right
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 Viya System Options: Reference.

Syntax

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

Syntax Description

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

Default 8
Range 2–20
Tip SAS writes a zero for a zero hour if the specified width is sufficient. For example, 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 \( \text{TOD}w.d \) 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 TODw.d format writes a leading zero for a single-hour digit. The TIMEw.d format and the HHMMw.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;</td>
<td>01:30</td>
</tr>
<tr>
<td>put begin tod5.;</td>
<td>01:30</td>
</tr>
<tr>
<td>begin = 1661437223;</td>
<td>14:20:23</td>
</tr>
<tr>
<td>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.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin = 32083;</td>
<td>08:54:43</td>
</tr>
<tr>
<td>put begin tod9.;</td>
<td>08:54:43</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “HHMM Format” on page 75
- “TIME Format” on page 98
- “TIMEAMPM Format” on page 100

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

Informats:
- “TIME Informat” on page 354
VAXRB Format

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

**Category:** Numeric  
**Alignment:** Right

**Syntax**

\[ \text{VAXRB} \quad w \cdot d \]

**Syntax Description**

- \( w \)
  - Specifies the width of the output field.
  - **Default:** 8
  - **Range:** 2-8

- \( d \)
  - Specifies the power of 10 by which to divide the value. This argument is optional.
  - **Default:** 0
  - **Range:** 0–31

**Details**

Use the VAXRB\(w,d\) format to write data in native VAX or VMS floating-point notation.

**Comparisons**

If you use SAS that is running under VAX or VMS, then the VAXRB\(w,d\) and the RB\(w,d\) formats are identical.

**Example**

```plaintext
x=1;
y=put(x,vaxrb8.);
put y=$hex16.;
```

<table>
<thead>
<tr>
<th>Value of ( x )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------------</td>
<td>--------</td>
</tr>
<tr>
<td>(--)0000000000000000</td>
<td>( \text{8040000000000000} )</td>
</tr>
</tbody>
</table>

* The result is the hexadecimal representation for the integer.
**w 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 Viya System Options: Reference.

---

### Syntax

```plaintext
w.d
```

### Syntax Description

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

- **Range**: 1–32  
- **Tip**: Allow enough space to write the value, the decimal point, and a minus sign, if necessary.

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

- **Range**: 0–31  
- **Requirement**: must be less than `w`  
- **Tip**: If `d` is 0 or you omit it, `w.d` writes the value without a decimal point.

---

### Details

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

---

### Comparisons

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

---

### Example

```plaintext
put @7 x 6.3;
```
### WEEKDATE Format

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

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

### Syntax

```
WEEKDATE w.
```

### Syntax Description

<table>
<thead>
<tr>
<th><code>w</code></th>
<th>Specifies the width of the output field.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>29</td>
</tr>
<tr>
<td><strong>Range</strong></td>
<td>3–37</td>
</tr>
</tbody>
</table>

### Details

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

- `dd` is an integer that represents the day of the month.
- `yy` or `yyyy` is a two-digit or four-digit integer that represents the year.

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

### Comparisons

The `WEEKDATE w.` format is the same as the `WEEKDATEx w.` format except that `WEEKDATEx w.` prints `dd` before the month's name.

### Example

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

WEEKDATX Format writes date values as the day of the week and date in the form day-of-week, dd month-name yy (or yyyy).

Syntax

WEEKDATXw.

See Also

Formats:
- “DATE Format” on page 141
- “DDMYY Format” on page 58
- “MMDDYY Format” on page 82
- “TOD Format” on page 102
- “WEEKDATX Format” on page 107
- “YYMMDD Format” on page 117

Functions:
- “JULDATE Function” in SAS Viya Functions and CALL Routines: Reference
- “MDY Function” in SAS Viya Functions and CALL Routines: Reference
- “WEEKDAY Function” in SAS Viya Functions and CALL Routines: Reference

Informats:
- “DATE Informat” on page 293
- “DDMYY Informat” on page 296
- “MMDDYY Informat” on page 318
- “YYMMDD Informat” on page 362
**Syntax Description**

\( w \)

specifies the width of the output field.

Default: 29

Range: 3–37

**Details**

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

*dd*

is an integer that represents the day of the month.

*yy* or *yyyy*

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

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

**Comparisons**

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

The *WEEKDATX*\( w \) format is the same as the *DTWKDATX*\( w \) format, except that *DTWKDATX*\( w \) expects a datetime value as input.

**Example**

The example table uses the input value of 19046, which is the SAS date value that corresponds to February 23, 2012.

<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:**

- “DATE Format” on page 141
- “DDMMYY Format” on page 58
- “DTWKDATX Format” on page 68
- “MMDDYY Format” on page 82
- “TOD Format” on page 102
- “WEEKDATE Format” on page 106
- “YYMMDD Format” on page 117
Functions:

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

Informats:

- “DATE Informat” on page 293
- “DDMMYY Informat” on page 296
- “MMDDYY Informat” on page 318
- “YYMMDD Informat” on page 362

**WEEKDAY Format**

Writes date values as the day of the week.

**Category:** Date and Time

**Alignment:** Right

**Syntax**

`WEEKDAY w.`

**Syntax Description**

`w` specifies the width of the output field.

- **Default:** 1
- **Range:** 1–32

**Details**

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

**Example**

The example table uses the input value of 19025, which is the SAS date value that corresponds to February 2, 2012.

<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>
**WEEKU 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:

<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 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,weeku3.);</td>
<td>2012-W05-03</td>
</tr>
<tr>
<td>w=put(sasdate,weeku5.);</td>
<td>2012-W05-03</td>
</tr>
<tr>
<td>x=put(sasdate,weeku7.);</td>
<td>2012-W05-03</td>
</tr>
<tr>
<td>y=put(sasdate,weeku9.);</td>
<td>2012-W05-03</td>
</tr>
<tr>
<td>z=put(sasdate,weeku11.);</td>
<td>2012-W05-03</td>
</tr>
<tr>
<td>put v;</td>
<td>12W05</td>
</tr>
<tr>
<td>put w;</td>
<td>12W05</td>
</tr>
<tr>
<td>put x;</td>
<td>12W05</td>
</tr>
<tr>
<td>put y;</td>
<td>2012W0503</td>
</tr>
<tr>
<td>put z;</td>
<td>2012-W05-03</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WEEKV Format” on page 111
- “WEEKW Format” on page 113

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

Informats:
- “WEEKU Informat” on page 394
- “WEEKV Informat” on page 396
- “WEEKW Informat” on page 398

WEEKV Format

WEEKV Format writes a week number in decimal format by using the V algorithm.

Category: Date and Time
Syntax

WEEKVw.

Syntax Description

w

specifies the width of the output field.

Default 11

Range 3–200

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>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. 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;
WEEKW Format

Writes a week number in decimal format by using the W algorithm.

**Syntax**

```
WEEKW w.
```

**Syntax Description**

`w`

specifies the width of the output field.

**Default** 11
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>Wwww</td>
<td>w01</td>
</tr>
<tr>
<td>5–6</td>
<td>yyWwww</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. 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

```sas
sasdate = '31JAN2012'd;
```

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>v=put(sasdate,weekw3.);</td>
<td>W05</td>
</tr>
<tr>
<td>w=put(sasdate,weekw5.);</td>
<td>12W05</td>
</tr>
<tr>
<td>x=put(sasdate,weekw7.);</td>
<td>12W0502</td>
</tr>
<tr>
<td>y=put(sasdate,weekw9.);</td>
<td>2012W0502</td>
</tr>
<tr>
<td>z=put(sasdate,weekw11.);</td>
<td>2012-W05-02</td>
</tr>
</tbody>
</table>
See Also

Formats:
- “WEEKU Format” on page 110
- “WEEKV Format” on page 111

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

Informats:
- “WEEKU Informat” on page 394
- “WEEKV Informat” on page 396
- “WEEKW Informat” on page 398

---

YEAR Format

Writes date values as the year.

**Category:** Date and Time

**Alignment:** Right

---

**Syntax**

\[ \text{YEAR}w. \]

**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>4</td>
<td>2–32</td>
<td>If ( w ) is less than 4, the last two digits of the year are printed. Otherwise, the year value is printed as four digits.</td>
</tr>
</tbody>
</table>

**Details**

The `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.
### YYMM 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.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>5–32</td>
</tr>
</tbody>
</table>

#### Interaction

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

#### Details

The `YYMMw` format writes SAS date values in the form `<yy>yyMmm`:

- `<yy>yy` is a two-digit or four-digit integer that represents the year.  
- `M` is the character separator to indicate that the number of the month follows.  
- `mm` is an integer that represents the month.
Example

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:
- “MMYY Format” on page 88
- “YYMMx Format” on page 226

YYMMDD 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

YYMMDDw.

Syntax Description

w

specifies the width of the output field.

- **Default:** 8
- **Range:** 2–10
Interaction

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

Details

The \( YYMMDDw \) format writes SAS date values in one of the following forms:

\[ yymmd \]  
\[ <yy>yy-mm-dd \]

where

\[ <yy>yy \]

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

\[ - \]

is the separator.

\[ mm \]

is an integer that represents the month.

\[ dd \]

is an integer that represents the day of the month.

To format a date that has a four-digit year and no separators, use the \( YYMMDDx \) format.

Example

The following examples use the input value of 19086, which is the SAS date value that corresponds to April 3, 2012.

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put day yymmd2.;</td>
<td>12</td>
</tr>
<tr>
<td>put day yymmd3.;</td>
<td>12</td>
</tr>
<tr>
<td>put day yymmd4.;</td>
<td>1204</td>
</tr>
<tr>
<td>put day yymmd5.;</td>
<td>12-04</td>
</tr>
<tr>
<td>put day yymmd6.;</td>
<td>120403</td>
</tr>
<tr>
<td>put day yymmd7.;</td>
<td>120403</td>
</tr>
<tr>
<td>put day yymmd8.;</td>
<td>12-04-03</td>
</tr>
<tr>
<td>put day yymmd10.;</td>
<td>2012-04-03</td>
</tr>
</tbody>
</table>
YYMMDDx Format

Writes date values in the form \textit{yyymmdd} or \textit{<yy>yy-mm-dd}, where the \textit{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.

\textbf{Category:} Date and Time

\textbf{Alignment:} Right

\textbf{Syntax}

\texttt{YYMMDD_{xw}}.

\textbf{Syntax Description}

\textit{x}

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

\begin{itemize}
  \item B \quad \text{separates with a blank.}
  \item C \quad \text{separates with a colon.}
  \item D \quad \text{separates with a hyphen.}
\end{itemize}
N indicates no separator.

P separates with a period.

S separates with a slash.

\( w \)

specifies the width of the output field.

Default 8

Range 2–10

**Interactions**

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

When \( x \) has a value of N, the width range is 2–8.

**Details**

The YYMDD\( xw \) format writes SAS date values in one of the following forms:

\( yymmdd \)

\(<yy>yyxmxdd\)

where

\(<yy>yy\)

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

\( x \)

is a specified separator.

\( mm \)

is an integer that represents the month.

\( dd \)

is an integer that represents the day of the month.

**Example**

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

<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 yymmddd8.;</td>
<td>12-05-14</td>
</tr>
<tr>
<td>put day yymmddp10.;</td>
<td>2012.05.14</td>
</tr>
</tbody>
</table>
YYMON Format

Writes date values in the form `ymmm` or `yyyyymmm`.

**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 `<yj>yymmm:`
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:*

- “MMYY Format” on page 88

**YYQ 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.

**Default** 6

**Range** 4–32

**Interaction** When `w` has a value of 4 or 5, the date appears with only the last two digits of the year. When `w` is 6 or more, the date appears with a four-digit year.
Details
The YYQw. format writes SAS date values in the form <yy>yyQq:

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

Q
is the character separator.

q
is an integer (1,2,3, or 4) that represents the quarter of the year.

Example
The following examples use the input value of 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:
- “YYQx Format” on page 123
- “YYQR Format” on page 125

YYQx 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.

| Default | 6 |
| Range   | 4–32 |

**Interactions**

When $x$ is set to $N$, no separator is specified. The width range is then 3–32, and the default changes to 5.

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

When $x$ has a value of $N$ and $w$ has a value of 3 or 4, the date appears with only the last two digits of the year. When $x$ has a value of $N$ and $w$ is 5 or more, the date appears with a four-digit year.

**Details**

The YYQ$xw.$ format writes SAS date values in one of the following forms:

\[ \langle \text{yy} \rangle \langle \text{yy} \rangle \langle \text{x} \rangle \langle q \rangle \]

\[ \langle \text{yy} \rangle \langle \text{yy} \rangle \langle \text{x} \rangle \langle q \rangle \]

where

\[ \langle \text{yy} \rangle \langle \text{yy} \rangle \]

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put date yyqc4.;</td>
<td>12:3</td>
</tr>
<tr>
<td>put date yyqd.;</td>
<td>2012-3</td>
</tr>
<tr>
<td>put date yyqn3.;</td>
<td>123</td>
</tr>
<tr>
<td>put date yyqp6.;</td>
<td>2012.3</td>
</tr>
<tr>
<td>put date yyqs8.;</td>
<td>2012/3</td>
</tr>
</tbody>
</table>

### See Also

**Formats:**
- “YYQ Format” on page 122
- “YYQRx Format” on page 126

### YYQR 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

**YYQR<w>**

**Syntax Description**

- `<w>` specifies the width of the output field.
  
  **Default**  8  
  **Range**  6-32  
  **Interaction**  When the value of <w> is too small to write a four-digit year, the date appears with only the last two digits of the year.

### Details

The YYQR<w> format writes SAS date values in the form `<yy>yyQqr`:

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

is the character separator.

$qr$

is a roman numeral (I, II, III, or IV) that represents the quarter of the year.

Example

The following examples use the input value of 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 yyqr8.;</td>
<td>2012QII</td>
</tr>
<tr>
<td>put date yyqr10.;</td>
<td>2012QII</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “YYQ Format” on page 122
- “YYQRx Format” on page 126

YYQRx 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.

<table>
<thead>
<tr>
<th>Default</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>6–32</td>
</tr>
</tbody>
</table>

**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<sup>xw</sup> format writes SAS date values in one of the following forms:

\[<yy>yyqr\]

\[<yy>yyxqr\]

where

\[<yy>yy\]

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

**x**  
is a specified separator.

**qr**  
is a roman numeral (I, II, III, or IV) that represents the quarter of the year.

**Example**

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

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>SAS Statement</td>
<td>Result</td>
</tr>
<tr>
<td>------------------</td>
<td>--------</td>
</tr>
<tr>
<td>put date yyqrc6.;</td>
<td>12:II</td>
</tr>
<tr>
<td>put date yyqrd.;</td>
<td>2012-II</td>
</tr>
<tr>
<td>put date yyqrn5.;</td>
<td>12II</td>
</tr>
<tr>
<td>put date yyqrp8.;</td>
<td>2012.II</td>
</tr>
<tr>
<td>put date yyqrs10.;</td>
<td>2012/II</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “YYQx Format” on page 123
- “YYQR Format” on page 125

**Z 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 Viya System Options: Reference.

**Syntax**

\[ Z^w.d \]

**Syntax Description**

\[ w \]

specifies the width of the output field.

**Default** 1

**Range** 1–32

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

\[ d \]

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

The $Zw.d$ format writes standard numeric values one digit per byte and fills in 0s to the left of the data value.

The $Zw.d$ format rounds to the nearest number that fits into the output field. If $w.d$ is too large to fit, SAS might shift the decimal to the BEST$w$. format. The $Zw.d$ format writes negative numbers with leading minus signs. In addition, it right aligns before writing and pads the output with leading zeros.

**Comparisons**

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

**Example**

```sas
   put @5 seqnum z8.;
```

<table>
<thead>
<tr>
<th>Value of seqnum</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>----1</td>
</tr>
<tr>
<td>1350</td>
<td>00001350</td>
</tr>
</tbody>
</table>
Chapter 4
Dictionary of Formats for SAS Only

Formats by Category .......................................................... 132

Dictionary ................................................................. 139
$ASCII Format ................................................................. 139
D Format ................................................................. 140
DATE Format ............................................................... 141
$EBCDIC Format ............................................................ 143
$MSGCASE Format .......................................................... 144
$N8601B Format ............................................................ 145
$N8601BA Format ........................................................... 146
$N8601E Format ............................................................. 147
$N8601EA Format ........................................................... 149
$N8601EH Format ........................................................... 150
$N8601EX Format ........................................................... 151
$N8601H Format ............................................................. 153
$N8601X Format ............................................................. 154
$VARyING Format .......................................................... 155
B8601DA Format ............................................................. 157
B8601DN Format ............................................................. 158
B8601DT Format ............................................................. 159
B8601DX Format ............................................................. 161
B8601DZ Format ............................................................. 162
B8601LX Format ............................................................. 164
B8601LZ Format ............................................................. 165
B8601TM Format ............................................................. 166
B8601TX Format ............................................................. 168
B8601TZ Format ............................................................. 169
E8601DA Format ............................................................. 171
E8601DN Format ............................................................. 172
E8601DT Format ............................................................. 173
E8601DX Format ............................................................. 174
E8601DZ Format ............................................................. 176
E8601LX Format ............................................................. 177
E8601LZ Format ............................................................. 179
E8601TM Format ............................................................. 180
E8601TX Format ............................................................. 182
E8601TZ Format ............................................................. 183
IB Format ................................................................. 185
IBR Format ................................................................. 186
JULDAY Format ............................................................. 188
NUMX Format ............................................................. 189
PD Format ................................................................. 190
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,</td>
</tr>
<tr>
<td></td>
<td>times, and datetimes.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>instructs SAS to write date, time, and datetime values using the ISO 8601</td>
</tr>
<tr>
<td></td>
<td>standard.</td>
</tr>
<tr>
<td>Numeric</td>
<td>instructs SAS to write numeric data values from numeric variables.</td>
</tr>
</tbody>
</table>

For information about formats that are valid in CAS, see Chapter 3, “Dictionary of Formats for SAS and the CAS Server,” on page 31. Formats that support national languages can be found in SAS Viya 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 *SAS Viya Utility Procedures Guide*.

The following table provides brief descriptions of the SAS formats that are supported in traditional SAS and CAS. 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>$ASCII Format (p. 139)</td>
<td>Converts native format character data to ASCII representation.</td>
</tr>
<tr>
<td></td>
<td>$EBCDIC Format (p. 143)</td>
<td>Converts native format character data to EBCDIC representation.</td>
</tr>
<tr>
<td></td>
<td>$MSGCASE Format (p. 144)</td>
<td>Writes character data in uppercase when the MSGCASE system option is in effect.</td>
</tr>
<tr>
<td></td>
<td>$VARYING Format (p. 155)</td>
<td>Writes character data of varying length.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>DATE Format (p. 141)</td>
<td>Writes date values in the form ddmmmyy, ddmmmyyyy, or dd-mm-yyyy.</td>
</tr>
<tr>
<td></td>
<td>$N8601B Format (p. 145)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyymmdTthmmss.</td>
</tr>
<tr>
<td></td>
<td>$N8601BA Format (p. 146)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PyyyymmddTthhmms and yyyymmdTthhmms.</td>
</tr>
<tr>
<td></td>
<td>$N8601E Format (p. 147)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyyy-mm-ddTth:mm:ss.</td>
</tr>
<tr>
<td></td>
<td>$N8601EA Format (p. 149)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddTth:mm:ss and yyyy-mm-ddTth:mm:ss.</td>
</tr>
<tr>
<td></td>
<td>$N8601EH Format (p. 150)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddTth:mm:ss and yyyy-mm-ddTth:mm:ss, using a hyphen (-) for omitted components.</td>
</tr>
<tr>
<td></td>
<td>$N8601EX Format (p. 151)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddTth:mm:ss and yyyy-mm-ddTth:mm:ss, using an x for each digit of an omitted component.</td>
</tr>
<tr>
<td></td>
<td>$N8601H Format (p. 153)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddTth:mm:ss, dropping omitted components in duration values and using a hyphen (-) for omitted components in datetime values.</td>
</tr>
<tr>
<td></td>
<td>$N8601X Format (p. 154)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddTth: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>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>B8601DA Format</td>
<td></td>
<td>Writes date values by using the ISO 8601 basic notation yyyymmd.</td>
</tr>
<tr>
<td>B8601DN Format</td>
<td></td>
<td>Writes dates from datetime values by using the ISO 8601 basic notation yyyymmd.</td>
</tr>
<tr>
<td>B8601DT Format</td>
<td></td>
<td>Writes datetime values by using the ISO 8601 basic notation yyyymmdThhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>B8601DX Format</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 by using the ISO 8601 datetime and time zone basic notation yyyymmdThhmmss+hhmm.</td>
</tr>
<tr>
<td>B8601DZ Format</td>
<td></td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone basic notation yyyymmdThhmmss+0000.</td>
</tr>
<tr>
<td>B8601LX Format</td>
<td></td>
<td>Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation yyyymmdThhmmss+</td>
</tr>
<tr>
<td>B8601LZ Format</td>
<td></td>
<td>Writes time values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation hhmmss+</td>
</tr>
<tr>
<td>B8601TM Format</td>
<td></td>
<td>Writes time values by using the ISO 8601 basic notation hhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>B8601TX Format</td>
<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 notation hhmmss+</td>
</tr>
<tr>
<td>B8601TZ Format</td>
<td></td>
<td>Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 basic time notation hhmmss+</td>
</tr>
<tr>
<td>E8601DA Format</td>
<td></td>
<td>Writes date values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DN Format</td>
<td></td>
<td>Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td>E8601DT Format</td>
<td></td>
<td>Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>E8601DX Format</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 by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+hh:mm.</td>
</tr>
<tr>
<td>E8601DZ Format</td>
<td></td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 datetime and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>E8601LX Format (p. 177)</td>
<td>Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss+</td>
<td>–hh:mm.</td>
</tr>
<tr>
<td>E8601LZ Format (p. 179)</td>
<td>Writes time values as local time, appending the Coordinated Universal Time (UTC) offset for the local SAS session, using the ISO 8601 extended time notation hh:mm:ss+</td>
<td>–hh:mm.</td>
</tr>
<tr>
<td>E8601TM Format (p. 180)</td>
<td>Writes time values by using the ISO 8601 extended notation hh:mm:ss.ffffff.</td>
<td></td>
</tr>
<tr>
<td>E8601TX Format (p. 182)</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>
<td>–hh:mm.</td>
</tr>
<tr>
<td>E8601TZ Format (p. 183)</td>
<td>Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation hh:mm:ss.&lt;fff&gt;+</td>
<td>–hh:mm.</td>
</tr>
<tr>
<td>JULDAY Format (p. 188)</td>
<td>Writes date values as the Julian day of the year.</td>
<td></td>
</tr>
<tr>
<td>PDJULG Format (p. 191)</td>
<td>Writes packed Julian date values in the hexadecimal format yyyydddF for IBM.</td>
<td></td>
</tr>
<tr>
<td>PDJULI Format (p. 193)</td>
<td>Writes packed Julian date values in the hexadecimal format ccyydddF for IBM.</td>
<td></td>
</tr>
<tr>
<td>WORDDATE Format (p. 221)</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>WORDDATX Format (p. 223)</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>YYMMx Format (p. 226)</td>
<td>Writes date values in the form &lt;yy&gt;yy:mm or &lt;yy&gt;yy-mm. The x in the format name represents the special character that separates the year and the month. This special character can be a hyphen (-), period (.), slash (/), colon (:), or no separator. The year can be either two or four digits.</td>
<td></td>
</tr>
<tr>
<td>ISO 8601</td>
<td>SN8601B Format (p. 145)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PnYnMnDTnHnMnS and yyyy-mm-ddThhmss.</td>
</tr>
<tr>
<td></td>
<td>SN8601BA Format (p. 146)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the basic notations PyyymmdThhmss and yyyy-mm-ddThhmss.</td>
</tr>
<tr>
<td></td>
<td>SN8601E Format (p. 147)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations PnYnMnDTnHnMnS and yyyy-mm-ddThhm:ss.</td>
</tr>
<tr>
<td></td>
<td>SN8601EA Format (p. 149)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThhm:ss and yyyy-mm-ddThhm:ss.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>$N8601EH Format (p. 150)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh:mm:ss, using a hyphen ( - ) for omitted components.</td>
<td></td>
</tr>
<tr>
<td>$N8601EX Format (p. 151)</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>
<td></td>
</tr>
<tr>
<td>$N8601H Format (p. 153)</td>
<td>Writes ISO 8601 duration, datetime, and interval forms PnYnMnDTnHnMnS and yyyy-mm-ddThh:mm:ss, dropping omitted components in duration values and using a hyphen ( - ) for omitted components in datetime values.</td>
<td></td>
</tr>
<tr>
<td>$N8601X Format (p. 154)</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>
<td></td>
</tr>
<tr>
<td>B8601DA Format (p. 157)</td>
<td>Writes date values by using the ISO 8601 basic notation yyyymmd.</td>
<td></td>
</tr>
<tr>
<td>B8601DN Format (p. 158)</td>
<td>Writes dates from datetime values by using the ISO 8601 basic notation yyyymmd.</td>
<td></td>
</tr>
<tr>
<td>B8601DT Format (p. 159)</td>
<td>Writes datetime values by using the ISO 8601 basic notation yyyymmdDTthmmss&lt;ffffff&gt;.</td>
<td></td>
</tr>
<tr>
<td>B8601DX Format (p. 161)</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 yyyymmdDTthmmss+hhmm.</td>
<td></td>
</tr>
<tr>
<td>B8601DZ Format (p. 162)</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 yyyymmdDTthmmss+0000.</td>
<td></td>
</tr>
<tr>
<td>B8601LX Format (p. 164)</td>
<td>Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 basic notation yyyymmdDTthmmss+</td>
<td>-hhmm.</td>
</tr>
<tr>
<td>B8601LZ Format (p. 165)</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>
<td>-hhmm.</td>
</tr>
<tr>
<td>B8601TM Format (p. 166)</td>
<td>Writes time values by using the ISO 8601 basic notation hhmmss&lt;ffffff&gt;.</td>
<td></td>
</tr>
<tr>
<td>B8601TX Format (p. 168)</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>B8601TZ Format (p. 169)</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>
</tbody>
</table>
## Formats by Category

<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>E8601DA Format (p. 171)</strong></td>
<td>Writes date values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
<td></td>
</tr>
<tr>
<td><strong>E8601DN Format (p. 172)</strong></td>
<td>Writes dates from SAS datetime values by using the ISO 8601 extended notation yyyy-mm-dd.</td>
<td></td>
</tr>
<tr>
<td><strong>E8601DT Format (p. 173)</strong></td>
<td>Writes datetime values by using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.ffffff.</td>
<td></td>
</tr>
<tr>
<td><strong>E8601DX Format (p. 174)</strong></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 date and time zone extended notation yyyy-mm-ddThh:mm:ss+hh:mm.</td>
<td></td>
</tr>
<tr>
<td><strong>E8601DZ Format (p. 176)</strong></td>
<td>Writes datetime values for the zero meridian Coordinated Universal Time (UTC) time by using the ISO 8601 date and time zone extended notation yyyy-mm-ddThh:mm:ss+00:00.</td>
<td></td>
</tr>
<tr>
<td><strong>E8601LX Format (p. 177)</strong></td>
<td>Writes datetime values as local time by appending a time zone offset difference between the local time and UTC, using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss+</td>
<td>–hh:mm.</td>
</tr>
<tr>
<td><strong>E8601LZ Format (p. 179)</strong></td>
<td>Writes time values as local time, appending the Coordinated Universal Time (UTC) offset for the local SAS session, using the ISO 8601 extended time notation hh:mm:ss+</td>
<td>–hh:mm.</td>
</tr>
<tr>
<td><strong>E8601TM Format (p. 180)</strong></td>
<td>Writes time values by using the ISO 8601 extended notation hh:mm:ss.ffffff.</td>
<td></td>
</tr>
<tr>
<td><strong>E8601TX Format (p. 182)</strong></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>
<td>–hh:mm.</td>
</tr>
<tr>
<td><strong>E8601TZ Format (p. 183)</strong></td>
<td>Adjusts time values to the Coordinated Universal Time (UTC) and writes the time values by using the ISO 8601 extended notation hh:mm:ss.&lt;fff&gt;+</td>
<td>–hh:mm.</td>
</tr>
<tr>
<td><strong>D Format (p. 140)</strong></td>
<td>Prints numeric values, possibly with a great range of values, lining up decimal places for values of similar magnitude.</td>
<td></td>
</tr>
<tr>
<td><strong>IB Format (p. 185)</strong></td>
<td>Writes native integer binary (fixed-point) values, including negative values.</td>
<td></td>
</tr>
<tr>
<td><strong>IBR Format (p. 186)</strong></td>
<td>Writes integer binary (fixed-point) values in Intel and DEC formats.</td>
<td></td>
</tr>
<tr>
<td><strong>NUMX Format (p. 189)</strong></td>
<td>Writes numeric values with a comma in place of the decimal point.</td>
<td></td>
</tr>
<tr>
<td><strong>PD Format (p. 190)</strong></td>
<td>Writes data in packed decimal format.</td>
<td></td>
</tr>
<tr>
<td><strong>PERCENT Format (p. 194)</strong></td>
<td>Writes numeric values as percentages.</td>
<td></td>
</tr>
<tr>
<td><strong>PERCENTN Format (p. 195)</strong></td>
<td>Produces percentages, using a minus sign for negative values.</td>
<td></td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>PIB Format (p. 197)</td>
<td>Writes positive integer binary (fixed-point) values.</td>
<td></td>
</tr>
<tr>
<td>PIBR Format (p. 198)</td>
<td>Writes positive integer binary (fixed-point) values in Intel and DEC formats.</td>
<td></td>
</tr>
<tr>
<td>PK Format (p. 200)</td>
<td>Writes data in unsigned packed decimal format.</td>
<td></td>
</tr>
<tr>
<td>PVALUE Format (p. 201)</td>
<td>Writes p-values.</td>
<td></td>
</tr>
<tr>
<td>RB Format (p. 202)</td>
<td>Writes real binary data (floating-point) in real binary format.</td>
<td></td>
</tr>
<tr>
<td>S370FF Format (p. 204)</td>
<td>Writes native standard numeric data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FIB Format (p. 205)</td>
<td>Writes integer binary (fixed-point) values, including negative values, in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FIBU Format (p. 206)</td>
<td>Writes unsigned integer binary (fixed-point) values in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPD Format (p. 208)</td>
<td>Writes packed decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPDU Format (p. 209)</td>
<td>Writes unsigned packed decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FPIB Format (p. 210)</td>
<td>Writes positive integer binary (fixed-point) values in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FRB Format (p. 212)</td>
<td>Writes real binary (floating-point) data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZD Format (p. 213)</td>
<td>Writes zoned decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDL Format (p. 214)</td>
<td>Writes zoned decimal leading–sign data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDS Format (p. 215)</td>
<td>Writes zoned decimal separate leading-sign data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDT Format (p. 216)</td>
<td>Writes zoned decimal separate trailing-sign data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>S370FZDU Format (p. 218)</td>
<td>Writes unsigned zoned decimal data in IBM mainframe format.</td>
<td></td>
</tr>
<tr>
<td>SSN Format (p. 219)</td>
<td>Writes Social Security numbers.</td>
<td></td>
</tr>
<tr>
<td>VMSZN Format (p. 220)</td>
<td>Generates VMS and MicroFocus COBOL zoned numeric data.</td>
<td></td>
</tr>
<tr>
<td>WORDF Format (p. 224)</td>
<td>Writes numeric values as words with fractions that are shown numerically.</td>
<td></td>
</tr>
<tr>
<td>WORDS Format (p. 225)</td>
<td>Writes numeric values as words.</td>
<td></td>
</tr>
<tr>
<td>ZD Format (p. 227)</td>
<td>Writes numeric data in zoned decimal format.</td>
<td></td>
</tr>
</tbody>
</table>
$ASCII 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 ASCII$w$.
- On all other systems, $ASCIIw$. behaves like the $CHARw$. format.

### Example

`put x $ascii3.;`

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result*</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>616263</td>
</tr>
<tr>
<td>ABC</td>
<td>414243</td>
</tr>
<tr>
<td>()</td>
<td>28293B</td>
</tr>
</tbody>
</table>

* 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.
**D Format**

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

**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 Viya System Options: Reference.

### Syntax

\[D_{w.p}\]

**Syntax Description**

\[w\]

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

- **Default:** 12
- **Range:** 1–32

\[p\]

specifies the precision. This argument is optional.

- **Default:** 3
- **Range:** 0–9
- **Requirement:** \( p \) must be less than \( w \)
- **Tips:** If \( p \) is omitted or is specified as 0, then \( p \) is set to 3.

If zero is the desired precision, use the \( w.d \) format in place of the \( D_{w.p} \) format.

### Details

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

### Comparisons

- The BEST\( w. \) format writes as many significant digits as possible in the output field, but if the numbers vary in magnitude, the decimal points do not line up.
- \( D_{w.p} \) writes numbers with the desired precision and more alignment than the BEST\( w. \) format.
• The 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

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

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

See Also

Formats:
• “BESTD Format” on page 48

DATE Format

W

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{ddmmyy}, 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.

\begin{tabular}{|l|l|}
\hline
SAS Statement & Result \\
\hline
put day date5. ; & 16MAR \\
put day date6. ; & 16MAR \\
put day date7. ; & 16MAR12 \\
put day date8. ; & 16MAR12 \\
put day date9. ; & 16MAR2012 \\
put day date11. ; & 16-MAR-2012 \\
\hline
\end{tabular}

See Also

Functions:

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

Informats:

- “DATE Informat” on page 293
$EBCDIC Format

Converts native format character data to EBCDIC representation.

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

Syntax
$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 DFLANG, DATESTYLE, and PAPERSIZE System Options Based on the LOCALE= System Option” in SAS Viya 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 Viya National Language Support (NLS): Reference Guide.

Comparisons

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

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>{Å}</td>
<td>043D0</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.

### $MSGCASE Format

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

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

#### Syntax

```sas
$MSGCASE
```

#### 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” in SAS Viya System Options: Reference

$N8601B Format

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

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

$SN8601B w.d$

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

- $PnYnMnDTnHnMnS$
- $yymmmdTthhmmss$
- $PnYnMnDTnHnMnS/yymmmdTthhmmss$
- $yymmmdTthhmmssT/PnYnMnDTnHnMnS$

The lowest-order component can contain fractions, as in these examples:
- $p2y3.5m$
- $p00020304T05.335$

**Example**

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

<table>
<thead>
<tr>
<th>Value of nb</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>000240505112FFC</td>
<td>P2Y4M5DT5H1M12S</td>
</tr>
<tr>
<td>2012915155300FFD</td>
<td>20120915T155300</td>
</tr>
<tr>
<td>2012915000000FFD201491500000FFD</td>
<td>20120915T000000/20140915T000000</td>
</tr>
<tr>
<td>0033104030255FFC2012915155300FFD</td>
<td>P33Y1M4DT3H2M5S/20120915T155300</td>
</tr>
</tbody>
</table>

**See Also**

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

---

**$N8601BA Format**

Writes ISO 8601 duration, datetime, and interval forms by using the basic notations `PyyymmddTthhmms` and `yymmddTthhmms`.

**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>Range</th>
<th>Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>1–200</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>
$d$
specifies the number of digits to the right of the lowest-order component. This argument is optional.

Default 0

Range 0–3

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

- $PyyyymmddThhmmss$
- $yyyymmddThhmmss$
- $PyyyymmddT hhmmss/yyyyymmdT hhmmss$
- $yyyyymddT hhmmss/ PyyyyymmdT hhmmss$

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>P00020405T050112.5</td>
</tr>
<tr>
<td>2012915155300FFD</td>
<td>20120915T155300</td>
</tr>
<tr>
<td>00023040506075282012915155300FFD</td>
<td>P00020304T050607.33/20120915T155300</td>
</tr>
</tbody>
</table>

See Also

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

$N8601E \_w,d$

Syntax Description

\_w
specifies the width of the output field.

Default 50
Range 1–200

\_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:

- $p2y3.5m$
- $p0002–03–04T05.335$

Example

put @1 ne $n8601e.;

<table>
<thead>
<tr>
<th>Value of ne</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00024050501127DD00</td>
<td>P2Y4M5DT5H1M12.5S</td>
</tr>
<tr>
<td>2012915155300FFD</td>
<td>2012-09-15T15:53:00</td>
</tr>
<tr>
<td>20129150000000FFD20149150000000FFD</td>
<td>2012-09-15T00:00/2013-09-15T00:00:00</td>
</tr>
<tr>
<td>0033104003255FFC2012915155300FFD</td>
<td>P33Y1M4DT3H2M5S/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

$N8601EA Format

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

Categories: Date and Time
ISO 8601

Alignment: Left

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.4.4, complete representation

Syntax

$N8601EAw.d

Syntax Description

$N8601EAw.d

\(w\)

specifies the width of the output field.

Default 50

Range 1–200

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

\(d\)

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

Default 0

Range 0–3

Details

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

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

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

- p00023.5
$N8601EH 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 a hyphen ( - ) for omitted components.

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

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

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

**Syntax**

$\$$N8601EHw.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
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>00023FFFFFFFFFFFFFC2012FFFF15FFFPFFFD</td>
<td>P0002-03--/2012-15:53:00</td>
</tr>
<tr>
<td>2012FFFF15FFFFFFFFddFFFFFF3FF1553FFFFFC</td>
<td>2012-03-15-15:53-00</td>
</tr>
</tbody>
</table>

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

$N8601EX Format
Writes ISO 8601 duration, datetime, and interval forms by using the extended notations Pyyyy-mm-ddThh:mm:ss and yyyy-mm-ddThh:mm:ss, using an x for each digit of an omitted component.

Categories: Date and Time
ISO 8601

Alignment: Left

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
$N8601EXw.d
**Syntax Description**

\( w \)

specifies the width of the output field.

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

\( d \)

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

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

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

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
$N8601H Format

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.

**Categories:** Date and Time
ISO 8601

**Alignment:** Left

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

**Supports:** ISO 8601 Elements 5.5.3, 5.5.4.1, and 5.5.4.2

---

**Syntax**

$N8601H w.d$

**Syntax Description**

$w$

specifies the width of the output field.

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

**Details**

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

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

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>0002304FFFFFFFFFC2012FFFF15FFFFFFFFD</td>
<td>P2Y3M4D/2012—T15:--:--</td>
</tr>
<tr>
<td>FFFFF102FFFFFFFFFD2012FFFF15FFFFFFFFD</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

$N8601X 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

```
$N8601X w.d
```

**Syntax Description**

- **\(w\)** specifies the width of the output field.
  - Default: 50
  - Range: 1–200
  - Requirement: The minimum length for a duration value or a datetime value is 16. The minimum length for an interval value is 16.

- **\(d\)** specifies the number of digits to the right of the lowest-order component. This argument is optional.
  - Default: 0
  - Range: 0–3
Details

The $N8601X format writes ISO 8601 durations, intervals, and datetimes in the following forms, omitting components in the \( PnYnMnDTnHnMnS \) form and using an \( x \) to represent omitted components in the datetime form:

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

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

$VARYING Format

Writes character data of varying length.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>in DATA step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>Character</td>
</tr>
<tr>
<td>Alignment:</td>
<td>Left</td>
</tr>
</tbody>
</table>

Syntax

$VARYING_w. length-variable

Syntax Description

\( w \)

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

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

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

Restriction

length-variable cannot be an array reference.

Requirement

You must specify length-variable immediately after $VARYING_w$ in a SAS statement.

Tips

If the value of length-variable is 0, negative, or missing, SAS writes nothing to the output field.

If the value of length-variable is greater than 0 but less than w, SAS writes the number of characters that are specified by length-variable.

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

Details

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

Examples

Example 1: Obtaining a Variable Length Directly

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

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

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

<table>
<thead>
<tr>
<th>Value of name</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>New York 8</td>
<td>New York</td>
</tr>
<tr>
<td>Toronto 7</td>
<td>Toronto</td>
</tr>
<tr>
<td>Buenos Aires 12</td>
<td>Buenos Aires</td>
</tr>
<tr>
<td>Tokyo 5</td>
<td>Tokyo</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.

---

B8601DA Format

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

- **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**

`B8601DAw`

**Syntax Description**

`w`

- specifies the width of the output field.

- **Default:** 10
- **Range:** 8–10
Details

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

- **yyyy**  
  is a four-digit year.

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

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

Example

```
put bda b8601da.;
```

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

B8601DN Format

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

**Categories:** Date and Time

**ISO 8601**

**Alignment:** Left

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

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

Syntax

```
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 \texttt{yyyyymmdd}:

\begin{itemize}
\item \texttt{yyyy} \\
\hspace{1cm} is a four-digit year.
\item \texttt{mm} \\
\hspace{1cm} is a two-digit month (zero padded) between 01 and 12.
\item \texttt{dd} \\
\hspace{1cm} is a two-digit day of the month (zero padded) between 01 and 31.
\end{itemize}

Example

\begin{verbatim}
put bdn b8601dn. ;
\end{verbatim}

\begin{tabular}{|l|l|}
\hline
Value of bdn & Result \\
\hline
1631664000 & 20110915 \\
\hline
\end{tabular}

See Also

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

\section*{B8601DT Format}

Writes datetime values by using the ISO 8601 basic notation \texttt{yyyyymmddThhmmss<ffffff>}. 

\begin{description}
\item[Categories:] Date and Time
\item[ISO 8601]
\item[Alignment:] Left
\item[Restriction:] UTC time zone offset values are not supported.
\item[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 Viya System Options: Reference.
\item[Supports:] ISO 8601 Element 5.4.1, complete representation
\end{description}

\section*{Syntax}

\texttt{B8601DT\textit{w}.\textit{d}}

\subsection*{Syntax Description}

\texttt{w} \\
\hspace{1cm} 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.

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

Example

\begin{verbatim}
put bdt b8601dt.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Value of bdt</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1631721180</td>
<td>20110915T155300</td>
</tr>
</tbody>
</table>

See Also

“Working with Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 14
B8601DX 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 basic notation \texttt{yyyyymmddThhmmss +hhmm}.

- **Categories**: Date and Time, ISO 8601
- **Alignment**: Left
- **Supports**: ISO 8601 Element 5.4.1, complete representation

**Syntax**

\texttt{B8601DXw.}

**Syntax Description**

\texttt{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:

- \texttt{yyyyymmddThhmmss+hhmm}

  - \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{+|--hhmm} is an hour and minute signed offset from zero meridian time. The offset must be \texttt{+|--hhmm} (that is, \texttt{+} or \texttt{–} and four characters).
Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

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

### Example

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

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

### See Also

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

---

**B8601DZ Format**

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

- **Categories:** Date and Time
- **ISO 8601**
- **Alignment:** Left
- **Supports:** ISO 8601 Element 5.4.1, complete representation

**Syntax**

`B8601DZw.`

**Syntax Description**

w

specifies the width of the output field.

Default 26
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:

- `yyyyymmddThhmmss+0000`  
  *Note:* Use this form when `w` is large enough to support this time zone notation.

- `yyyyymmddThhmmssZ`  
  *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 “B8601LZ Format” on page 165.

**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

<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>
</tbody>
</table>

```
put bdz b8601dz20.;
```
## Datetime Value | Value of bdz | Result
---|---|---
20110915T155300Z | 1631721180 | 20110915T155300+0000

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

### See Also

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

### B8601LX Format

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

**Categories:** Date and Time  
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:

- `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**

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

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

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

**See Also**

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

**B8601LZ 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 *hhmss*+–*hhmm*.

**Categories:** Date and Time

ISO 8601

**Alignment:** Left

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

**Syntax**

*B8601LZw.*

**Syntax Description**

*w* 

specifies the width of the output field.
Details

The B8601LZ format writes time values without making any adjustments, and appends the UTC time zone offset for the local SAS session by using the ISO 8601 basic notation _hhmmss_±_hhmm_:

- **hh**
  - is a two-digit hour (zero padded) between 00 and 23.
- **mm**
  - is a two-digit minute (zero padded) between 00 and 59.
- **ss**
  - is a two-digit second (zero padded) between 00 and 59.
- **±hhmm**
  - is an hour and minute signed offset from zero meridian time. Note that the offset must be ±_hhmm_ (that is, + or – and four characters).

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

**Restriction:** The shorter form ±_hh_ is not supported.

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

Example

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

```
put blz b8601lz.;
```

<table>
<thead>
<tr>
<th>Value of blz</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>46380</td>
<td>125300-0500</td>
</tr>
</tbody>
</table>

See Also

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

---

**B8601TM Format**

Writes time values by using the ISO 8601 basic notation _hhmmss<fff_.

**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 Viya System Options: Reference.

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

## Syntax

B8601TM<sub>w.d</sub>

### Syntax Description

- **w** specifies the width of the output field.
  - Default: 8
  - Range: 6–15

- **d** specifies the number of digits to the right of the seconds value that represents a fraction of a second. This argument is optional.
  - Default: 0
  - Range: 0–6

### Details

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

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

### Example

```sas
put btm b8601tm.;
```
Value of btm | Result
---|---
57180 | 155300

**See Also**

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

**B8601TX 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 `hhmmss+|–hhmm`.

**Categories:** Date and Time

ISO 8601

**Alignment:** Right

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

**Syntax**

\[ \text{B8601TX}_w. \]

**Syntax Description**

\[ w \]

specifies the width of the output field.

**Default** 14

**Range** 9–20

**Details**

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

- \[ hhmmss+|–hhmm \]

  \[ hh \]

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

  \[ mm \]

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

  \[ ss \]

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

  \[ +|–hhmm \]

  is an hour and minute signed offset from zero meridian time. The offset must be +|–hhmm (that is, + or – and four characters).
Use + for time zones east of the zero meridian, and use – for time zones west of the zero meridian. For example, +0200 indicates a two-hour time difference to the east of the zero meridian, and –0600 indicates a six-hour time difference to the west of the zero meridian.

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

When SAS reads a UTC time by using the B8601TZ informat, and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 000000 and 240000. If the B8601TX format attempts to format a time outside of this time range, the time is formatted with asterisks to indicate that the value is out of range.

**Example**

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

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

**See Also**

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

---

**B8601TZ Format**

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

**Categories:** Date and Time

**ISO 8601**

**Alignment:** Left

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

**Syntax**

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:

- \[ hh:mm:ss \pm hh:mm \]
  
  *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 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

```plaintext
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

E8601DA 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

```sas
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

---

**E8601DN Format**

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

**Categories:** Date and Time

**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.

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

- `yyyy` is a four-digit year.
- `mm` is a two-digit month (zero padded) between 01 and 12.
- `dd` is a two-digit day of the month (zero padded) between 01 and 31.
Example

```
put edn e8601dn.;
```

<table>
<thead>
<tr>
<th>Value for edn</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>166308532</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

---

**E8601DT 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 Viya System Options: Reference.
- **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.
  - Default: 0
  - Range: 0–6
Details

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

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

Example

```sas
put edt e8601dt 25.3.;
```

<table>
<thead>
<tr>
<th>Value of edt</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1663343580.2</td>
<td>2012-09-15T15:53:00.234</td>
</tr>
</tbody>
</table>

See Also

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

E8601DX 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`.

**Categories:** Date and Time  
ISO 8601  

**Alignment:** Left  

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

**Syntax**

```sas
E8601DXw:
```
Syntax Description

\( w \)

specifies the width of the output field.

Default    26
Range       20–35

Details
UTC values specify a date and time that are based on the zero meridian in Greenwich, England. Using this format, SAS converts a datetime value to the UTC value and determines the user local date and time by using the value of the TIMEZONE= system option. If the TIMEZONE= option is not set, the user local date and time are based on the local date and time. The E8601DX format writes SAS datetime values by using 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>2013-02-01T07:34:56-05:00</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>
E8601DZ Format

WritessdatetimevaluesforthezeromeridianCoordinatedUniversalTime(UTC)timebyusingtheISO
8601datetimeandtimezonelongnotations:yyyy-mm-ddThh:mm:ss±00:00.

**Categories:** Date and Time

**Alignment:** Left

**Alias:** IS8601DZw.

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

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

### Syntax

E8601DZw.

### Syntax Description

w

specifiesthewidthoftheoutputfield.

**Default** 26

**Range** 20–35

### Details

UTCvaluesspecifyatimeandatimezonebasedonthezeromeridianinGreenwich,
England. The E8601DZ format writes SAS datetime values by using one of the
following ISO 8601 extended datetime notations:

- yyyy-mm-ddThh:mm:ss±00:00
  
  **Note:** Use this form when w is large enough to support this time zone notation.

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

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

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

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

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

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

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

An ISO 8601 time or datetime value that specifies a time zone offset is adjusted by the number of hours and minutes that is specified in the offset and processed as the time or datetime for the zero meridian (Greenwich, England). The E8601DZ format always writes the datetime value by using the zero meridian offset value of +00:00. To write a datetime that uses the UTC offset other than +00:00, see “E8601LZ Format” on page 179.

Restriction: The shorter form +00 is not supported.

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

Example

```
put edz e8601dz.;
```

<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

---

E8601LX 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

E8601LX\(w\)

Syntax Description

\(w\)

specifies the width of the output field.

Default 26

Range 20–35

Details

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

- \(\text{yyyy-mm-ddThh:mm:ss+hh:mm}\)

  \(\text{yyyy}\)

  is a four-digit year.

  \(\text{mm}\)

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

  \(\text{dd}\)

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

  \(\text{hh}\)

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

  \(\text{mm}\)

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

  \(\text{ss}\)

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

  \(+\text{--hh:mm}\)

  is an hour and minute signed offset from zero meridian time. The offset must be \(+\text{--hh:mm}\) (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 \(+\text{--hh}\) is not supported.

Example

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

```sas
blx='01Feb2013T12:34:56'\text{dt};
put blx e8601lx.;
```
### E8601LZ Format

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

**Categories:** Date and Time  
ISO 8601  

**Alignment:** Left  

**Alias:** 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:

```sas
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

---

**E8601TM Format**

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

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

**Alignment:** Left
Alias: IS8601TMw.d

Restriction: UTC time zone offset values are not supported.

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

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

Syntax

E8601TMw.d

Syntax Description

w

specifies the width of the output field.

Default 8

Range 8–15

d

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

Default 0

Range 0–6

Details

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

hh

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

mm

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

ss

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

ffffff

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

Example

put etm e8601tm.;

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

E8601TX 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 $hh:mm:ss\pm hh:mm$.

Categories: Date and Time  
ISO 8601

Alignment: Right

Supports: ISO 8601 Elements 5.3.3 and 5.3.4

Syntax

E8601TX$w$.

Syntax Description

$w$

specifies the width of the output field.

Default 14

Range 9–20

Details

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

- $hh:mm:ss\pm hh:mm$
  - $hh$ 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.
  - $\pm hh:mm$ 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, $+0200$ 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>
</table>
| data _null_;  
t='12:34:56't;  
put t e8601tx.;  
run; | 07:34:56-05:00 |
| options timezone='America/Adak';  
data _null_;  
t='12:34:56't;  
put t e8601tx.;  
run; | 02:34:56-10:00 |

**See Also**

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

---

**E8601TZ 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 Viya System Options: Reference.

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

**Syntax**

E8601TZw.d
Syntax Description

\textit{w}

specifies the width of the output field.

Default 14

Range 9–20

\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

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:

• \textit{hh:mm:ss.<fff>+|--hh:mm}

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

• \textit{hh:mm:ssZ}

\textit{Note:} Use this form when \textit{w} is not large enough to support the \textit{+|--hh:mm} time zone notation.

\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{fff}

are optional fractional seconds.

\textit{+|--hh:mm}

is an hour and minute signed offset from zero meridian time. The offset must be \textit{+|--hh:mm} (that is, + or – and five characters).

\textit{Restriction:} The shorter form \textit{+|--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.

\textit{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

IB Format

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

**Category:** Numeric

**Alignment:** Left

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

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

```
y=put(x,ib4.);
put y $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>----+----1</td>
<td>----+----1</td>
</tr>
<tr>
<td>128</td>
<td>00000080</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:
- “IBR Format” on page 186

IBR Format

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

Category: Numeric
Alignment: Left
Syntax

\texttt{IBR}w.d

\textbf{Syntax Description}

\textit{w}

specifies the width of the output field.

\begin{itemize}
  \item Default: 4
  \item Range: 1–8
\end{itemize}

\textit{d}

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

\begin{itemize}
  \item Default: 0
  \item Range: 0–10
\end{itemize}

\section*{Details}

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

\textbf{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.

\section*{Comparisons}

\begin{itemize}
  \item The \texttt{IB}\textit{w}.\textit{d} and \texttt{PIB}\textit{w}.\textit{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.)
  \item The \texttt{IBR}\textit{w}.\textit{d} and \texttt{PIBR}\textit{w}.\textit{d} formats are used to write little endian integers, regardless of the operating environment that you are writing on.
  \item In Intel and DEC operating environments, the \texttt{IB}\textit{w}.\textit{d} and \texttt{IBR}\textit{w}.\textit{d} formats are equivalent.
\end{itemize}

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.

\section*{Example}

\begin{verbatim}
y=put(x,ibr4.);
put y $hex8.;
\end{verbatim}
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:
- “IB Format” on page 185

JULDAY Format

Writess date values as the Julian day of the year.

Category: Date and Time
Alignment: Right

Syntax

JULDAYw.

Syntax Description

w
specifies the width of the output field.

Default 3
Range 3–32

Details

The JULDAYw. 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></td>
<td>80000000</td>
</tr>
</tbody>
</table>
NUMX 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 Viya 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 NUMXw.d format writes numeric values with a comma in place of the decimal point.

### Comparisons

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

### Example: Examples

```sas
put x numx10.2;
```
### PD 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 Viya System Options: Reference.

### Syntax

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

### Syntax Description

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result *</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.

PDJULG 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 **PDJULG** \( w \) 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**

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>date = '17mar2012'd;</td>
<td>2012077F</td>
</tr>
<tr>
<td>juldate = put(date,pdjulg4.);</td>
<td></td>
</tr>
<tr>
<td>put juldate $hex8.;</td>
<td></td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**
- “**JULDAY Format**” on page 188
- “**JULIAN Format**” on page 80
- “**PDJULI Format**” on page 193

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

**Informat:**
- “**JULIAN Informat**” on page 315
- “**PDJULG Informat**” on page 389
- “**PDJULI Informat**” on page 390
System Options:

- “YEARCUTOFF= System Option” in SAS Viya System Options: Reference

PDJULI Format

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

Category: Date and Time

Syntax

\text{PDJULI}w.

Syntax Description

\textit{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:

\textit{cc}

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

\textit{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.

\textit{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).

\textit{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></td>
<td>-1-----+-----</td>
</tr>
</tbody>
</table>

PDJULI Format  193
SAS Statement | Result
---|---
date = '17mar2012'd;
juldate = put(date,pdjuli4.);
put juldate $hex8.; | 0112077F

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

See Also

Formats:
- “JULDAY Format” on page 188
- “JULIAN Format” on page 80
- “PDJULG Format” on page 191

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

Informat:
- “JULIAN Informat” on page 315
- “PDJULG Informat” on page 389
- “PDJULI Informat” on page 390

System Options:
- “YEARCUTOFF= System Option” in SAS Viya System Options: Reference

PERCENT 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 Viya 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 PERCENT\( w.d \) format multiplies values by 100, formats them the same as the BEST\( w.d \) format, and adds a percent sign (%) to the end of the formatted value. Negative values are enclosed in parentheses.

**Example**

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

<table>
<thead>
<tr>
<th>Value of ( x )</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.05</td>
<td>(</td>
</tr>
<tr>
<td>-0.1</td>
<td>10%</td>
</tr>
<tr>
<td>1.2</td>
<td>120%</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “PERCENTN Format” on page 195

---

**PERCENTN Format**

Produces percentages, using a minus sign for negative values.

**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 Viya System Options: Reference.

**Syntax**

PERCENTN_{w.d}

**Syntax Description**

\( w \)

specifies the width of the output field.

- **Default**: 6
- **Range**: 4–32
- **Tip**: The width of the output field must account for the minus sign ( - ), the percent sign ( % ), and a trailing blank, whether the number is negative or positive.

\( d \)

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

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

**Details**

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

**Comparisons**

The PERCENTN_{w.d} format produces percents by using a minus sign instead of parentheses for negative values. The 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>
</tbody>
</table>
Value of \( x \) | Result
---|---
-0.05 | -5%
-6.3 | -630%

**See Also**

**Format:**
- “PERCENT Format” on page 194

**PIB Format**

Writes positive integer binary (fixed-point) values.

**Category:** Numeric

**Alignment:** Left

**Syntax**

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

**Syntax Description**

\( w \)

specifies the width of the output field.

Default 1

Range 1–8

\( d \)

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

Default 0

Range 0–31

**Details**

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

*Note:* Different operating environments store integer binary values in different ways. This concept is called byte ordering. For 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\textit{w.d} format treats all values as positive and includes the sign bit as part of the value.

• The PIB\textit{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\textit{w.d} format is the same as the IB\textit{w.d} format except that PIB\textit{w.d} treats all values as positive values.

• The IB\textit{w.d} and PIB\textit{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\textit{w.d} and PIBR\textit{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

\begin{verbatim}
y=put(x,pib1.);
put y $hex2.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Value of \textit{x}</th>
<th>Result \textit{y}</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:

• “PIBR Format” on page 198

PIBR 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 PIBR\(w.d\) format treats all values as positive and includes the sign bit as part of the value.
- The PIBR\(w.d\) format with a width of 1 results in a value that corresponds to the binary equivalent of the contents of a byte. A value that corresponds to the binary equivalent of the contents of a byte is useful if your data contain values between hexadecimal 80 and hexadecimal FF, where the high-order bit can be misinterpreted as a negative sign.
- On Intel and DEC operating environments, the PIB\(w.d\) and PIBR\(w.d\) formats are equivalent.
- The IB\(w.d\) and PIB\(w.d\) formats are used to write native format integers. (Native format enables you to read and write values that are created in the same operating environment.) The IBR\(w.d\) and PIBR\(w.d\) formats are used to write little endian integers in any operating environment.

To view a table that shows the type of format to use with big endian and little endian integers, see “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

\[
y = \text{put}(x, \text{pibr2.});
\]
\[
\text{put } y \text{ } \$\text{hex4.};
\]

<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:

- “PIB Informat” on page 326

PK 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 Viya System Options: Reference.

Syntax

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

**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 PKw.d format is similar to the PDw.d format except that PKw.d does not write the sign in the low-order byte.

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</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.

PVALUE 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 Viya System Options: Reference.

Syntax

\[ PVALUE^{w.d} \]

**Syntax Description**

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

- **d**
  - specifies the number of digits to the right of the decimal point in the numeric value.
  - This argument is optional.
  - Default: the minimum of 4 and \( w-2 \)
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></td>
<td>----+-----1</td>
</tr>
<tr>
<td>.05</td>
<td>0.0500</td>
</tr>
<tr>
<td>0.000001</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>0</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>.0123456</td>
<td>0.0123</td>
</tr>
</tbody>
</table>

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

Category: Numeric
Alignment: Left

Syntax

RBw.d

Syntax Description

w
specifies the width of the output field.

Default: 4

Range: 2–8
The `RBw.d` format writes numeric data in the same way that SAS stores them. Because it requires no data conversion, `RBw.d` is the most efficient method for writing data with SAS.

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

**CAUTION:**

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

### Comparisons

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

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>RB4.</td>
<td>RB8.</td>
</tr>
<tr>
<td>Fortran</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>double</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-1</td>
<td>COMP-2</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>E</td>
<td>D</td>
</tr>
</tbody>
</table>

### Example

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<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.
S370FF Format

Writes native standard numeric data in IBM mainframe format.

Syntax

\texttt{S370FFw.d}

Syntax Description

- \textit{w} specifies the width of the output field.
  - Default: 12
  - Range: 1–32

- \textit{d} specifies the power of 10 by which to divide the value. This argument is optional.
  - Range: 0–31

Details

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

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

Comparisons

- On an EBCDIC system, S370FF\textit{w.d} behaves like the \textit{w.d} format.
- On all other systems, S370FF\textit{w.d} performs the same role for numeric data that the \$EBCDIC\textit{w.} format does for character data.

Example

\begin{verbatim}
y=put(x,s370ff5.);
put y $hex10.;
\end{verbatim}

<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.
S370FIB Format

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

**Category:** Numeric

**Alignment:** Left

**Syntax**

\[
\text{S370FIB} \, w.d
\]

**Syntax Description**

\(w\)

specifies the width of the output field.

- **Default:** 4
- **Range:** 1–8

\(d\)

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

- **Default:** 0
- **Range:** 0–10

**Details**

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

Use S370FIB\(w.d\) to write integer binary data in IBM mainframe format from data that are created in other operating environments.

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

**Comparisons**

- If you use SAS on an IBM mainframe, S370FIB\(w.d\) and IB\(w.d\) 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

```sas
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:
• “S370FIBU Format” on page 206
• “S370FPFIB Format” on page 210

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

Use S370FIBU$w.d$ to write unsigned integer binary data in IBM mainframe format from data that are created in other operating environments.

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

**Comparisons**

- The S370FIBU$w.d$ format is equivalent to the COBOL notation PIC 9($n$) BINARY, where $n$ is the number of digits.
- The S370FIBU$w.d$ format is the same as the S370FIB$w.d$ format except that the S370FIBU$w.d$ format always uses the absolute value instead of the signed value.
- The S370FPIB$w.d$ format writes all negative numbers as FFs, while the S370FIBU$w.d$ format writes the absolute value.
- S370FPIB$w.d$, S370FIBU$w.d$, and S370FIB$w.d$ are used to write big endian integers in any operating environment.

To view a table that shows the type of format to use with big endian and little endian integers, see “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,s370fibu1.);
p 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.
S370FPD 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 Viya System Options: Reference.

Syntax

S370FPD<var/>w.d

Syntax Description

<var/>w  
specifies the width of the output field.  
Default 1  
Range 1–16

<var/>d  
specifies to multiply the number by 10<var/>. This argument is optional.  
Default 0  
Range 0–31

Details

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

Comparisons

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

<table>
<thead>
<tr>
<th>Language</th>
<th>Packed Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FPD4.</td>
</tr>
</tbody>
</table>
Language | Packed Decimal Notation
---|---
PL/I | FIXED DEC(7,0)
COBOL | COMP-3 PIC S9(7)
IBM 370 assembler | PL4

### Example

```snippets
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.

### S370FPDU 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 Viya 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.
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 S370FP Dw.d format except that the S370FP Dw.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

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result $hex4.</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.

S370FPIB 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.

  **Default:** 4

  **Range:** 1–8
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,s370fpib1.);
pay 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:
S370FRB Format

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

**Category:** Numeric

**Alignment:** Left

### Syntax

\[
\text{S370FRB} w.d
\]

### Syntax Description

- **w**
  - Specifies the width of the output field.
  - **Default:** 4
  - **Range:** 2–8

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

### Details

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

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

### Comparisons

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

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FRB4</td>
<td>S370FRB8</td>
</tr>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
<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>
IBM 370 assembler

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM 370 assembler</td>
<td>E</td>
<td>D</td>
</tr>
</tbody>
</table>

C

float
double

**Example**

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

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>42800000</td>
</tr>
<tr>
<td>-123</td>
<td>C2800000</td>
</tr>
</tbody>
</table>

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

---

### S370FZD 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 Viya System Options: Reference.

### Syntax

**S370FZD**<sub>\(w,d\)</sub>

**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 S370FZDw.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.);
put 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 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 Viya System Options: Reference.
Syntax

S370FZDLw.d

Syntax Description

w
specifies the width of the output field.

Default 8
Range 1–32

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

Default 0
Range 0–31

Details

Use S370FZDLw.d in other operating environments to write zoned decimal leading-sign data in the same format as on an IBM mainframe computer.

Comparisons

• The S370FZDLw.d format is similar to the S370FZDw.d format except that the S370FZDLw.d format displays the sign of the number in the first byte of the formatted output.

• The S370FZDLw.d format is equivalent to the COBOL notation PIC S9(n) DISPLAY SIGN LEADING, where the n value is the number of digits.

Example

y=put(x,s370fzd13.);
put y $hex6.;

<table>
<thead>
<tr>
<th>Value of x</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>C1F2F3</td>
</tr>
<tr>
<td>-123</td>
<td>D1F2F3</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.

S370FZDS Format

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

Category: Numeric
Syntax
S370FZDS\textsubscript{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
\[
y=\text{put} (x, \text{s370fzds}4.);
\text{put} y \text{ $hex8.};
\]

<table>
<thead>
<tr>
<th>Value of (x)</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>4EF1F2F3</td>
</tr>
<tr>
<td>-123</td>
<td>60F1F2F3</td>
</tr>
</tbody>
</table>

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

S370FZDT 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 Viya 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.
S370FZDU 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 Viya 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

```sas
y=put (x,s370fzdu3.);
puy $hex6. ;
```
The result is a hexadecimal representation of a binary number in zoned decimal format on an IBM mainframe computer. Each pair of hexadecimal characters (such as F1) corresponds to one byte of binary data, and each byte corresponds to one column of the output field.

**SSN 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>263878439</td>
<td>263-87-8439</td>
</tr>
</tbody>
</table>
VMSZN Format

Generates VMS and MicroFocus COBOL zoned numeric data.

**Category:** Numeric

**Alignment:** Left

### Syntax

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

### 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 VMSZNw.d format is similar to the ZDw.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>
</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:**
- “ZD Format” on page 227

**Informats:**
- “VMSZN Informat” on page 359

---

**WORDDATE 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 worddate1.;</td>
<td>Jun</td>
</tr>
<tr>
<td>put term worddate9.;</td>
<td>June</td>
</tr>
<tr>
<td>put term worddate12.;</td>
<td>Jun 14, 2012</td>
</tr>
<tr>
<td>put term worddate20.;</td>
<td>June 14, 2012</td>
</tr>
</tbody>
</table>

See Also

Formats:

- “WORDDATX Format” on page 223
WORDDATX Format

Writes date values as the day, the name of the month, and the year in the form \textit{dd month-name yyyy}.

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

### Syntax

\texttt{WORDDATXw}.

### Syntax Description

\texttt{w} specifies the width of the output field.

- **Default:** 18
- **Range:** 3–32

### Details

The \texttt{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 \texttt{WORDDATXw} format is the same as the \texttt{WORDDATEw} format except that \texttt{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:
WORDF Format

Writes numeric values as words with fractions that are shown numerically.

**Category:** Numeric

**Alignment:** Left

### Syntax

```
WORDF w.
```

### Syntax Description

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

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>5–32767</td>
</tr>
</tbody>
</table>

### Details

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

### Example

```
put price wordf15.;
```

<table>
<thead>
<tr>
<th>Value of price</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.5</td>
<td>two and 50/100</td>
</tr>
</tbody>
</table>

### See Also

Formats:
WORDS Format

Write numeric values as words.

Category: Numeric
Alignment: Left

Syntax

`WORDS w`.

Syntax Description

`w` specifies the width of the output field.

Default: 10
Range: 5–32767

Details

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

Negative numbers are preceded by the word minus. If the number is not an integer, the fractional portion is represented as hundredths. For example, 5.3 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 `WORDS w` format is similar to the `WORDF w` format except that `WORDS w` prints fractions as words instead of numbers.

Example

```
put price words23.;
```

<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:
YYMMx Format

Writes date values in the form <yy>yymm or <yy>yy-mm. The x in the format name represents the special character that separates the year and the month. This special character can be a hyphen (-), period (.), slash (/), colon (:), or no separator. The year can be either two or four digits.

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

---

**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 YYMM$xw$ format writes SAS date values in one of the following forms:

$<$yy$>$yy$xmm$

$<$yy$>$yy$Xmm$

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 yymn4.;</td>
<td>1205</td>
</tr>
<tr>
<td>put date yymnp8.;</td>
<td>2012.05</td>
</tr>
<tr>
<td>put date yymns10.;</td>
<td>2012/05</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “MMYYx Format” on page 89
- “YYMM Format” on page 116
information, see “DECIMALCONV= System Option” in SAS Viya System Options: Reference.

Syntax

\[ ZD_{w.d} \]

**Syntax Description**

\( w \)

specifies the width of the output field.

Default: 1

Range: 1–32

\( d \)

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

Default: 0

Range: 0–31

**Details**

The zoned decimal format is similar to standard numeric format in that every digit requires one byte. However, the value's sign is in the last byte, along with the last digit.

*Note:* Different operating environments store zoned decimal values in different ways. However, the \(ZD_{w,d}\) format writes zoned decimal values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

**Comparisons**

The following table compares the zoned decimal format with notation in several programming languages:

<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**

```sas
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.
Chapter 5
Overview of Informats

Definition of Informats .......................... 233
Syntax ............................................. 234
Using Informats .................................. 234
   Ways to Specify Informats ...................... 234
   Permanent versus Temporary Association ... 236
   User-Defined Informats ...................... 236
Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms .................................. 237
   Definitions ................................... 237
   How the Bytes Are Ordered .................... 237
   Reading Data Generated on Big Endian or Little Endian Platforms .............. 238
   Integer Binary Notation in Different Programming Languages ................. 238
Working with Packed Decimal and Zoned Decimal Data .................. 239
   Definitions ................................... 239
   Types of Data ................................ 240
   Platforms Supporting Packed Decimal and Zoned Decimal Data ................ 241
   Languages Supporting Packed Decimal and Zoned Decimal Data ................ 241
   Summary of Packed Decimal and Zoned Decimal Formats and Informats .......... 242
Reading Dates and Times By Using the ISO 8601 Basic and Extended Notations . 243
   ISO 8601 Formatting Symbols ................. 243
   Reading ISO 8601 Date, Time, and Datetime Values ...................... 244
   Reading ISO 8601 Duration, Interval, and Datetime Values .................. 246

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:

```syntax
<$>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 *SAS Viya Utility 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.

When a problem occurs with an informat, SAS writes a note to the SAS log and assigns a missing value to the variable. Problems occur if you use an incompatible informat, such as a numeric informat to read character data, or if you specify the width of a date and time informat that causes SAS to read a special character in the last column.

**Using Informats**

**Ways to Specify Informats**

**Overview of Specifying Informats**

You can specify informats in the following ways:

- in an INPUT statement
• with the INPUT, INPUTC, and INPUTN functions
• in an INFORMAT statement in a DATA step or a PROC step
• in an ATTRIB statement in a DATA step or a PROC step

**INPUT Statement**
The INPUT statement with an informat after a variable name is the simplest way to read values into a variable. For example, the following INPUT statement uses two informats:

```
input @15 style $3. @21 price 5.2;
```

The $w. character informat reads values into the variable STYLE. The w.d numeric informat reads values into the variable PRICE.

For a complete discussion of the INPUT statement, see “INPUT Statement” in *SAS Viya Statements: Reference*.

**INPUT Function**
The INPUT function converts a SAS character expression using a specified informat. The informat determines whether the resulting value is numeric or character. Thus, the INPUT function is useful for converting data. Here is an example:

```
TempCharacter='98.6';
TemperatureNumber=input(TempCharacter,4.);
```

The INPUT function in combination with the w.d informat converts the character value of TempCharacter to a numeric value and assigns the numeric value 98.6 to TemperatureNumber.

**TIP** If the first argument of the INPUT function is a character expression and not a variable, the character expression must include quotation marks.

Use the PUT function with a SAS format to convert numeric values to character values. For an example of a numeric-to-character conversion, see “PUT Function” in *SAS Viya Functions and CALL Routines: Reference*. For a complete discussion of the INPUT function, see “INPUT Function” in *SAS Viya Functions and CALL Routines: Reference*.

**INFORMAT Statement**
The INFORMAT statement associates an informat with a variable. SAS uses the informat in any subsequent INPUT statement to read values into the variable. For example, in the following statements the INFORMAT statement associates the DATE w. informat with the variables Birthdate and Interview:

```
informat Birthdate Interview date9.;
input @63 Birthdate Interview;
```

An informat that is associated with an INFORMAT statement behaves like an informat that you specify with a colon (:) format modifier in an INPUT statement. For details about using the colon (:) modifier, see “INPUT Statement, List” in *SAS Viya 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 Viya 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:

```sas
attrib Birthdate Interview informat=date9.;
input @63 Birthdate Interview;
```

An informat that is associated by using the INFORMAT= option in the ATTRIB statement behaves like an informat that you specify with a colon (:) format modifier in an INPUT statement. For details about using the colon (:) modifier, see “INPUT Statement, List” in *SAS Viya 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 Viya 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 *SAS Viya Utility 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>
</table>
| FMTERR        | SAS produces an error that causes the current DATA or
               | PROC step to stop. |
| NOFMTERR      | SAS continues processing by substituting a default
               | informat. |

Although using NOFMTERR enables SAS to process a variable, you lose the
information that the user-defined informat supplies. This option can cause a DATA step
to misread data, and it can produce incorrect results. For more information, see

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: HP-UX, AIX, Solaris on SPARC, and
Macintosh. In SAS, the following platforms are considered little endian: Intel ABI,
Linux, OpenVMS, and OpenVMS Integrity.

**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, FE FF,
and FE FF FF FF. These representations result from the output of the integer binary
value –2 expressed in hexadecimal representation.
Reading Data Generated on Big Endian or Little Endian Platforms

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

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

<table>
<thead>
<tr>
<th>Platform for Which the Data Was Created</th>
<th>Platform the Data Is Read On</th>
<th>Signed Integer</th>
<th>Informat</th>
</tr>
</thead>
<tbody>
<tr>
<td>big endian</td>
<td>big endian</td>
<td>yes</td>
<td>IB or S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>big endian</td>
<td>no</td>
<td>PIB, S370FPIB, S370FIBU</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>big endian</td>
<td>little endian</td>
<td>no</td>
<td>PIBR</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>big endian</td>
<td>no</td>
<td>PIBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>yes</td>
<td>IB or IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>little endian</td>
<td>no</td>
<td>PIB or PIBR</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>yes</td>
<td>S370FIB</td>
</tr>
<tr>
<td>big endian</td>
<td>either</td>
<td>no</td>
<td>S370FPIB</td>
</tr>
<tr>
<td>little endian</td>
<td>either</td>
<td>yes</td>
<td>IBR</td>
</tr>
<tr>
<td>little endian</td>
<td>either</td>
<td>no</td>
<td>PIBR</td>
</tr>
</tbody>
</table>

Integer Binary Notation in Different Programming Languages

The following table compares integer binary notation according to programming language.
### Working with Packed Decimal and Zoned Decimal Data

**Definitions**

Packed decimal

specifies a method of encoding decimal numbers by using each byte to represent two decimal digits. Packed decimal representation stores decimal data with exact precision. The fractional part of the number is determined by the informat or format because there is no separate mantissa and exponent.

An advantage of using packed decimal data is that exact precision can be maintained. However, computations involving decimal data might become inexact due to the lack of native instructions.

Zoned decimal

specifies a method of encoding decimal numbers in which each digit requires 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.

<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.
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 $\text{ceil}\left(\frac{x+1}{2}\right)$. For PIC 9(X) PACKED-DECIMAL, W is $\text{ceil}\left(\frac{x}{2}\right)$. For example, PIC S9(5) PACKED-DECIMAL represents five digits. If a sign is included, six nibbles are needed. $\text{ceil}\left(\frac{5+1}{2}\right)$ has a length of 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</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>0001D</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 Informats

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

<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>
</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 / 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

  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>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>hhmmssZ</td>
<td>155300Z</td>
<td>B8601TZw.d</td>
<td></td>
</tr>
<tr>
<td>Convert to local time with a time zone offset</td>
<td>hhmmss+-hhmm</td>
<td>155300+0500</td>
<td>B8601TZw.d</td>
</tr>
<tr>
<td>Datetime with no time zone offset</td>
<td>yyyyymmddThhmmssnnnnn</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>yyyyymmddThhmmssZ</td>
<td>20130915T155300Z</td>
<td>B8601DZw.d</td>
<td></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>Extended Notations</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 reads ISO datetime, duration, and interval values.

$N8601B informat
- reads duration, interval, and datetime values that are specified in either the basic notation or the extended notation

$N8601E informat
- reads duration, interval, and datetime values that are specified only in the extended notation

Use the $N8601E informat when you want to make sure that you are in compliance with the extended notation.

The datetime values that are read by these informats result in a SAS character representation. If you want a datetime value to be read as a numeric value, use the B8601DT informat, the B8601DZ informat, the E8601DT informat, or the E8601DZ informat.

**Complete Duration, Interval, and Datetime Notations**

The following table shows the formatting of duration, datetime, and interval values that can be read in the complete form:

<table>
<thead>
<tr>
<th>Time Component</th>
<th>ISO 8601 Notation</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration - Basic Notation</td>
<td>PYYYYMMDDThhmss</td>
<td>P20120915T155300</td>
</tr>
<tr>
<td></td>
<td>–PYYYYMMDDThhmss</td>
<td>–P20080915T155300</td>
</tr>
<tr>
<td>Duration - Extended Notation</td>
<td>PYYYY-MM-DDThh:mm:ss</td>
<td>P2012-09-15T15:53:00</td>
</tr>
<tr>
<td>Time Component</td>
<td>ISO 8601 Notation</td>
<td>Example</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td></td>
<td>–PYYYY-MM-DDThh:mm:ss</td>
<td>–P2012-09-15T15:53:00</td>
</tr>
<tr>
<td>Duration - Basic and Extended Notation</td>
<td>PnYnMnDTnHnMnS</td>
<td>P2y10m14dT20h13m45s</td>
</tr>
<tr>
<td></td>
<td>–PnYnMnDTnHnMnS</td>
<td>–P2n10m14dT20h13m45s</td>
</tr>
<tr>
<td></td>
<td>PnW (weeks)</td>
<td>P6w</td>
</tr>
<tr>
<td></td>
<td>YYYYMMDDDTthhmmss/</td>
<td>20120915T155300/20141113</td>
</tr>
<tr>
<td>Interval - Basic Notation</td>
<td>YYYYMMDDDTthhmmss</td>
<td>T000000</td>
</tr>
<tr>
<td></td>
<td>PnYnMnDTnHnMnS/ YYYYMMDDDTthhmmss</td>
<td>P2y10M14dT20h13m45s/20120915T155300</td>
</tr>
<tr>
<td></td>
<td>YYYYMMDDDTthhmmss/PnYnMnDTnHnMnS</td>
<td>20120915T155300/P2y10M14dT20h13m45s</td>
</tr>
<tr>
<td>Interval- Extended Notation</td>
<td>YYYY-MM-DDThh:mm:ss/ YYYY-MM-DDThh:mm:ss</td>
<td>2012-09-15T15:53:00/2014-11-13T00:00</td>
</tr>
<tr>
<td></td>
<td>PnYnMnDTnHnMnS/ YYYY-MM-DDThh:mm:ss</td>
<td>P2y10M14dT20h13m45s/2012-09-15T15:53:00</td>
</tr>
<tr>
<td></td>
<td>YYYY-MM-DDThh:mm:ss/PnYnMnDTnHnMnS</td>
<td>2012-09-15T15:53:00/P2y10M14dT20h13m45s</td>
</tr>
<tr>
<td>Datetime-Basic Notation</td>
<td>YYYYYMMDDDTthhmmss.fff</td>
<td>20120915T155300</td>
</tr>
<tr>
<td></td>
<td>+</td>
<td>–hhmm</td>
</tr>
<tr>
<td></td>
<td>(all blank)</td>
<td></td>
</tr>
<tr>
<td>Datetime-Extended Notation</td>
<td>YYYY-MM-DDThh:mm:ss.fff+</td>
<td>–hhmm</td>
</tr>
<tr>
<td></td>
<td>(all blank)</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 YYYYMMDD. SAS reads omitted components using the $N8601B informat or the $N8601E informat, and the omitted component must be represented by a hyphen (-).

The following examples show duration, datetime, and interval values with omitted components:

- **p0003 - 02 -T10:31:33**
  - The omitted component is the number of days.

- **–p0003 - 02 -T- :31:33**
  - The omitted component is the number of hours.
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</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:

- 201209.5
- P2012-09-15T10.33
- 2012-09-15/P0003-03-03,333
Chapter 6
Informats Documented in Other Publications

### Chapter 7

Dictionary of Informats for SAS and the CAS Server

<table>
<thead>
<tr>
<th>Informats by Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>254</td>
</tr>
<tr>
<td>$ASCII Informat</td>
<td>260</td>
</tr>
<tr>
<td>$BASE64X Informat</td>
<td>261</td>
</tr>
<tr>
<td>$BINARY Informat</td>
<td>262</td>
</tr>
<tr>
<td>$CB Informat</td>
<td>263</td>
</tr>
<tr>
<td>$CHAR Informat</td>
<td>264</td>
</tr>
<tr>
<td>$CHARZB Informat</td>
<td>265</td>
</tr>
<tr>
<td>$EBCDIC Informat</td>
<td>266</td>
</tr>
<tr>
<td>$HEX Informat</td>
<td>267</td>
</tr>
<tr>
<td>$N8601B Informat</td>
<td>268</td>
</tr>
<tr>
<td>$N8601E Informat</td>
<td>270</td>
</tr>
<tr>
<td>$OCTAL Informat</td>
<td>272</td>
</tr>
<tr>
<td>$PHEX Informat</td>
<td>273</td>
</tr>
<tr>
<td>$QUOTE Informat</td>
<td>274</td>
</tr>
<tr>
<td>$UPCASE Informat</td>
<td>275</td>
</tr>
<tr>
<td>$VARYING Informat</td>
<td>276</td>
</tr>
<tr>
<td>$w Informat</td>
<td>278</td>
</tr>
<tr>
<td>ANYDTDTE Informat</td>
<td>279</td>
</tr>
<tr>
<td>ANYDTDTM Informat</td>
<td>281</td>
</tr>
<tr>
<td>ANYDTTME Informat</td>
<td>285</td>
</tr>
<tr>
<td>BINARY Informat</td>
<td>287</td>
</tr>
<tr>
<td>BITS Informat</td>
<td>288</td>
</tr>
<tr>
<td>BZ Informat</td>
<td>289</td>
</tr>
<tr>
<td>CB Informat</td>
<td>290</td>
</tr>
<tr>
<td>COMMA Informat</td>
<td>291</td>
</tr>
<tr>
<td>COMMAX Informat</td>
<td>292</td>
</tr>
<tr>
<td>DATE Informat</td>
<td>293</td>
</tr>
<tr>
<td>DATETIME Informat</td>
<td>295</td>
</tr>
<tr>
<td>DDMMYY Informat</td>
<td>296</td>
</tr>
<tr>
<td>E8601DA Informat</td>
<td>298</td>
</tr>
<tr>
<td>E8601DN Informat</td>
<td>299</td>
</tr>
<tr>
<td>E8601DT Informat</td>
<td>300</td>
</tr>
<tr>
<td>E8601DX Informat</td>
<td>302</td>
</tr>
<tr>
<td>E8601DZ Informat</td>
<td>302</td>
</tr>
<tr>
<td>E8601TM Informat</td>
<td>304</td>
</tr>
<tr>
<td>E8601TZ Informat</td>
<td>305</td>
</tr>
<tr>
<td>FLOAT Informat</td>
<td>307</td>
</tr>
<tr>
<td>HEX Informat</td>
<td>308</td>
</tr>
<tr>
<td>HHMMSS Informat</td>
<td>309</td>
</tr>
<tr>
<td>IB Informat</td>
<td>311</td>
</tr>
</tbody>
</table>
There are five categories of informats in this list:

<table>
<thead>
<tr>
<th>Category</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYMMN Informat</td>
<td>322</td>
</tr>
<tr>
<td>VMSZN Informat</td>
<td>317</td>
</tr>
<tr>
<td>TODSTAMP Informat</td>
<td>323</td>
</tr>
<tr>
<td>TIME Informat</td>
<td>324</td>
</tr>
<tr>
<td>SMFSTAMP Informat</td>
<td>325</td>
</tr>
<tr>
<td>S370FRB Informat</td>
<td>326</td>
</tr>
<tr>
<td>S370FPDU Informat</td>
<td>327</td>
</tr>
<tr>
<td>S370FIB Informat</td>
<td>328</td>
</tr>
<tr>
<td>ZDV Informat</td>
<td>329</td>
</tr>
<tr>
<td>ZD Informat</td>
<td>330</td>
</tr>
<tr>
<td>PERCENT Informat</td>
<td>331</td>
</tr>
<tr>
<td>ZDV Informat</td>
<td>332</td>
</tr>
<tr>
<td>YYMM Informat</td>
<td>333</td>
</tr>
<tr>
<td>ZD Informat</td>
<td>334</td>
</tr>
<tr>
<td>YYQ Informat</td>
<td>335</td>
</tr>
<tr>
<td>YYMMN Informat</td>
<td>336</td>
</tr>
<tr>
<td>ZD Informat</td>
<td>337</td>
</tr>
<tr>
<td>PIB Informat</td>
<td>338</td>
</tr>
<tr>
<td>PIBR Informat</td>
<td>339</td>
</tr>
<tr>
<td>PIBR Informat</td>
<td>340</td>
</tr>
<tr>
<td>RMFDUR Informat</td>
<td>341</td>
</tr>
<tr>
<td>RMFDUR Informat</td>
<td>342</td>
</tr>
<tr>
<td>S370FZDB Informat</td>
<td>343</td>
</tr>
<tr>
<td>S370FZDL Informat</td>
<td>344</td>
</tr>
<tr>
<td>S370FZDS Informat</td>
<td>345</td>
</tr>
<tr>
<td>S370FZDT Informat</td>
<td>346</td>
</tr>
<tr>
<td>S370FZDU Informat</td>
<td>347</td>
</tr>
<tr>
<td>SHRSTAMP Informat</td>
<td>348</td>
</tr>
<tr>
<td>SMFSTAMP Informat</td>
<td>349</td>
</tr>
<tr>
<td>TIME Informat</td>
<td>350</td>
</tr>
<tr>
<td>TODSTAMP Informat</td>
<td>351</td>
</tr>
<tr>
<td>TRAILSIGN Informat</td>
<td>352</td>
</tr>
<tr>
<td>TU Informat</td>
<td>353</td>
</tr>
<tr>
<td>VAXRB Informat</td>
<td>354</td>
</tr>
<tr>
<td>VMSZSN Informat</td>
<td>355</td>
</tr>
<tr>
<td>w Informat</td>
<td>356</td>
</tr>
<tr>
<td>YYMMDD Informat</td>
<td>357</td>
</tr>
<tr>
<td>YYMMN Informat</td>
<td>358</td>
</tr>
<tr>
<td>YYQ Informat</td>
<td>359</td>
</tr>
<tr>
<td>ZD Informat</td>
<td>360</td>
</tr>
<tr>
<td>ZDB Informat</td>
<td>361</td>
</tr>
<tr>
<td>ZDV Informat</td>
<td>362</td>
</tr>
<tr>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Character</td>
<td>instructs SAS to read character data values into character variables.</td>
</tr>
<tr>
<td>Column Binary</td>
<td>instructs SAS to read data stored in column-binary or multipunched form into character or numeric variables.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>instructs SAS to read date values into variables that represent dates, times, and datetimes.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>instructs SAS to read date, time, and datetime values that are written in the ISO 8601 standard into either numeric or character variables.</td>
</tr>
<tr>
<td>Numeric</td>
<td>instructs SAS to read numeric data values into numeric variables.</td>
</tr>
</tbody>
</table>

Informats created with PROC FORMAT are supported on CAS. For information about creating user-defined informats, see “FORMAT Procedure” in SAS Viya Utility 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>$ASCII Informat (p. 260)</td>
<td>Converts ASCII character data to native format.</td>
</tr>
<tr>
<td></td>
<td>$BASE64X Informat (p. 261)</td>
<td>Converts ASCII text into character data by using Base 64 encoding.</td>
</tr>
<tr>
<td></td>
<td>$BINARY Informat (p. 262)</td>
<td>Converts binary data to character data.</td>
</tr>
<tr>
<td></td>
<td>$CHAR Informat (p. 264)</td>
<td>Reads character data with blanks.</td>
</tr>
<tr>
<td></td>
<td>$CHARZB Informat (p. 265)</td>
<td>Converts binary 0s to blanks.</td>
</tr>
<tr>
<td></td>
<td>$EBCDIC Informat (p. 266)</td>
<td>Converts EBCDIC character data to native format.</td>
</tr>
<tr>
<td></td>
<td>$HEX Informat (p. 267)</td>
<td>Converts hexadecimal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$OCTAL Informat (p. 272)</td>
<td>Converts octal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$PHEX Informat (p. 273)</td>
<td>Converts packed hexadecimal data to character data.</td>
</tr>
<tr>
<td></td>
<td>$QUOTE Informat (p. 274)</td>
<td>Removes matching quotation marks from character data.</td>
</tr>
<tr>
<td></td>
<td>$UPCASE Informat (p. 275)</td>
<td>Converts character data to uppercase.</td>
</tr>
<tr>
<td></td>
<td>$VARYING Informat (p. 276)</td>
<td>Reads character data of varying length.</td>
</tr>
<tr>
<td>Column Binary</td>
<td>SCB Informat (p. 263)</td>
<td>Reads standard character data from column-binary files.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CB Informat (p. 290)</td>
<td></td>
<td>Reads standard numeric values from column-binary files.</td>
</tr>
<tr>
<td>PUNCH Informat (p. 331)</td>
<td></td>
<td>Reads whether a row of column-binary data is punched.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>SN8601B Informat (p. 268)</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>SN8601E Informat (p. 270)</td>
<td>Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.</td>
</tr>
<tr>
<td></td>
<td>ANYDTDTE Informat (p. 279)</td>
<td>Reads and extracts the date value from various date, time, and datetime forms.</td>
</tr>
<tr>
<td></td>
<td>ANYDTDTM Informat (p. 281)</td>
<td>Reads and extracts datetime values from various date, time, and datetime forms.</td>
</tr>
<tr>
<td></td>
<td>ANYDTTME Informat (p. 285)</td>
<td>Reads and extracts time values from various date, time, and datetime forms.</td>
</tr>
<tr>
<td></td>
<td>DATE Informat (p. 293)</td>
<td>Reads date values in the form ddmmmyy or ddmmmyyyyy.</td>
</tr>
<tr>
<td></td>
<td>DATETIME Informat (p. 295)</td>
<td>Reads datetime values in the form ddmmmyy hh:mm:ss.ss or ddmmmyyyyy hh:mm:ss.</td>
</tr>
<tr>
<td></td>
<td>DDMMYY Informat (p. 296)</td>
<td>Reads date values in the form ddmmmyy&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></td>
<td>E8601DA Informat (p. 298)</td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.</td>
</tr>
<tr>
<td></td>
<td>E8601DN Informat (p. 299)</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>E8601DT Informat (p. 300)</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>E8601DZ Informat (p. 302)</td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime extended notation yyyy-mm-ddThh:mm:ss.&lt;ffffff&gt; or yyyy-mm-ddThh:mm:ss.&lt;ffffff&gt;–Z.</td>
</tr>
<tr>
<td></td>
<td>E8601TM Informat (p. 304)</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>E8601TZ Informat (p. 305)</td>
<td>Reads time values that are specified using the ISO 8601 extended time notation hh:mm:ss&lt;ffffff&gt; or hh:mm:ssZ.</td>
</tr>
<tr>
<td></td>
<td>HHMMSS Informat (p. 309)</td>
<td>Reads hours, minutes, and seconds in the form hh:mm:ss or hhmmss.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>JULIAN Informat (p. 315)</td>
<td>Reads Julian dates in the form yyddd or yyyymddd.</td>
<td></td>
</tr>
<tr>
<td>MDYAMPM Informat (p. 316)</td>
<td>Reads datetime values in the form mm-dd-yy&lt;yy&gt; 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.</td>
<td></td>
</tr>
<tr>
<td>MMDDYY Informat (p. 318)</td>
<td>Reads date values in the form mmddyy or mmddyyyy.</td>
<td></td>
</tr>
<tr>
<td>MONYY Informat (p. 319)</td>
<td>Reads month and year date values in the form mmmyy or mmmyyyy.</td>
<td></td>
</tr>
<tr>
<td>MSEC Informat (p. 321)</td>
<td>Reads TIME MIC values.</td>
<td></td>
</tr>
<tr>
<td>PDTIME Informat (p. 325)</td>
<td>Reads packed decimal time of SMF and RMF records.</td>
<td></td>
</tr>
<tr>
<td>RMFDUR Informat (p. 333)</td>
<td>Reads duration intervals of RMF records.</td>
<td></td>
</tr>
<tr>
<td>RMFSSTAMP Informat (p. 334)</td>
<td>Reads time and date fields of RMF records.</td>
<td></td>
</tr>
<tr>
<td>SHRSTAMP Informat (p. 352)</td>
<td>Reads date and time values of SHR records.</td>
<td></td>
</tr>
<tr>
<td>SMFSSTAMP Informat (p. 353)</td>
<td>Reads time and date values of SMF records.</td>
<td></td>
</tr>
<tr>
<td>TIME Informat (p. 354)</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>
<td></td>
</tr>
<tr>
<td>TODSTAMP Informat (p. 356)</td>
<td>Reads an eight-byte time-of-day stamp.</td>
<td></td>
</tr>
<tr>
<td>TU Informat (p. 358)</td>
<td>Reads timer units.</td>
<td></td>
</tr>
<tr>
<td>YYMMDD Informat (p. 362)</td>
<td>Reads date values in the form yymmdd or yyyymddd.</td>
<td></td>
</tr>
<tr>
<td>YYMMN Informat (p. 364)</td>
<td>Reads date values in the form yyyyymm or yymm.</td>
<td></td>
</tr>
<tr>
<td>YYQ Informat (p. 365)</td>
<td>Reads quarters of the year in the form yyQq or yyyyQq.</td>
<td></td>
</tr>
<tr>
<td>ISO 8601</td>
<td>$N8601B Informat (p. 268)</td>
<td>Reads complete, truncated, and omitted forms of ISO 8601 duration, datetime, and interval values that are specified in either the basic or extended notations.</td>
</tr>
<tr>
<td>$N8601E Informat (p. 270)</td>
<td>Reads ISO 8601 duration, datetime, and interval values that are specified in the extended notation.</td>
<td></td>
</tr>
<tr>
<td>E8601DA Informat (p. 298)</td>
<td>Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.</td>
<td></td>
</tr>
<tr>
<td>E8601DN Informat (p. 299)</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>
<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>E8601DT Informat (p. 300)</td>
<td>Reads datetime values that are specified using the ISO 8601 extended notation yyyy-mm-ddThh:mm:ss.</td>
</tr>
<tr>
<td></td>
<td>E8601DZ Informat (p. 302)</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>E8601TM Informat (p. 304)</td>
<td>Reads time values that are specified using the ISO 8601 extended notation hh:mm:ss.</td>
</tr>
<tr>
<td></td>
<td>E8601TZ Informat (p. 305)</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>BINARY Informat (p. 287)</td>
<td>Converts positive binary values to integers.</td>
</tr>
<tr>
<td></td>
<td>BITS Informat (p. 288)</td>
<td>Extracts bits.</td>
</tr>
<tr>
<td></td>
<td>BZ Informat (p. 289)</td>
<td>Converts blanks to 0s.</td>
</tr>
<tr>
<td></td>
<td>COMMA Informat (p. 291)</td>
<td>Removes embedded characters.</td>
</tr>
<tr>
<td></td>
<td>COMMAX Informat (p. 292)</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>FLOAT Informat (p. 307)</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>HEX Informat (p. 308)</td>
<td>Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values.</td>
</tr>
<tr>
<td></td>
<td>IB Informat (p. 311)</td>
<td>Reads native integer binary (fixed-point) values, including negative values.</td>
</tr>
<tr>
<td></td>
<td>IBR Informat (p. 312)</td>
<td>Reads integer binary (fixed-point) values in Intel and DEC formats.</td>
</tr>
<tr>
<td></td>
<td>IEEE Informat (p. 313)</td>
<td>Reads an IEEE floating-point value and divides it by 10 raised to the dth power.</td>
</tr>
<tr>
<td></td>
<td>NUMX Informat (p. 322)</td>
<td>Reads numeric values with a comma in place of the decimal point.</td>
</tr>
<tr>
<td></td>
<td>PD Informat (p. 323)</td>
<td>Reads data that are stored in IBM packed decimal format.</td>
</tr>
<tr>
<td></td>
<td>PERCENT Informat (p. 326)</td>
<td>Reads percentages as numeric values.</td>
</tr>
<tr>
<td></td>
<td>PIB Informat (p. 326)</td>
<td>Reads positive integer binary (fixed-point) values.</td>
</tr>
<tr>
<td></td>
<td>PIBR Informat (p. 328)</td>
<td>Reads positive 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>PK Informat (p. 330)</td>
<td></td>
<td>Reads unsigned packed decimal data.</td>
</tr>
<tr>
<td>RB Informat (p. 332)</td>
<td></td>
<td>Reads numeric data that is stored in real binary (floating-point) notation.</td>
</tr>
<tr>
<td>S370FF Informat (p. 336)</td>
<td></td>
<td>Reads EBCDIC numeric data.</td>
</tr>
<tr>
<td>S370FIB Informat (p. 337)</td>
<td></td>
<td>Reads integer binary (fixed-point) values, including negative values, in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FIBU Informat (p. 338)</td>
<td></td>
<td>Reads unsigned integer binary (fixed-point) values in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPD Informat (p. 340)</td>
<td></td>
<td>Reads packed data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPDU Informat (p. 341)</td>
<td></td>
<td>Reads unsigned packed decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FPIB Informat (p. 342)</td>
<td></td>
<td>Reads positive integer binary (fixed-point) values in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FRB Informat (p. 344)</td>
<td></td>
<td>Reads real binary (floating-point) data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZD Informat (p. 345)</td>
<td></td>
<td>Reads zoned decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDB Informat (p. 346)</td>
<td></td>
<td>Reads zoned decimal data in which zeros have been left blank.</td>
</tr>
<tr>
<td>S370FZDL Informat (p. 347)</td>
<td></td>
<td>Reads zoned decimal leading-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDS Informat (p. 349)</td>
<td></td>
<td>Reads zoned decimal separate leading-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDT Informat (p. 350)</td>
<td></td>
<td>Reads zoned decimal separate trailing-sign data in IBM mainframe format.</td>
</tr>
<tr>
<td>S370FZDU Informat (p. 351)</td>
<td></td>
<td>Reads unsigned zoned decimal data in IBM mainframe format.</td>
</tr>
<tr>
<td>TRAILSGN Informat (p. 357)</td>
<td></td>
<td>Reads a trailing plus (+) or minus (–) sign.</td>
</tr>
<tr>
<td>VAXRB Informat (p. 359)</td>
<td></td>
<td>Reads real binary (floating-point) data in VMS format.</td>
</tr>
<tr>
<td>VMSZN Informat (p. 359)</td>
<td></td>
<td>Reads VMS and MicroFocus COBOL zoned numeric data.</td>
</tr>
<tr>
<td>w Informat (p. 361)</td>
<td></td>
<td>Reads standard numeric data.</td>
</tr>
<tr>
<td>ZD Informat (p. 367)</td>
<td></td>
<td>Reads zoned decimal data.</td>
</tr>
<tr>
<td>ZDB Informat (p. 368)</td>
<td></td>
<td>Reads zoned decimal data in which zeros have been left blank.</td>
</tr>
<tr>
<td>ZDV Informat (p. 369)</td>
<td></td>
<td>Reads and validates zoned decimal data.</td>
</tr>
</tbody>
</table>
Dictionary

$ASCII 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**

```
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>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.
$BASE64X Informat

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

**Category:** Character

**Alignment:** Left

---

**Syntax**

$BASE64Xw.

**Syntax Description**

w

specifies the width of the input field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1–32767</td>
</tr>
</tbody>
</table>

**Details**

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

Here are some uses of Base 64 encoding:

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

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

**Example**

```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>d3d3Lm15ZG9tYWiuLmNvbi9teWhpZGRlVlVSTA==</td>
<td><a href="http://www.mydomain.com/myhiddenURL">www.mydomain.com/myhiddenURL</a></td>
</tr>
</tbody>
</table>
## $BINARY Informat

Converts binary data to character data.

**Category:** Character

### Syntax

$$BINARY w.$$  

### Syntax Description

$w$ specifies the width of the input field. Because eight bits of binary information represent one character, every eight characters of input that $BINARY w.$ reads becomes one character value stored in a variable.

If $w < 8$, $BINARY w.$ reads the data as $w$ characters followed by 0s. Thus, $BINARY 4.$ reads the characters 0101 as 01010000, which converts to an EBCDIC & or an ASCII P. If $w > 8$ but is not a multiple of 8, $BINARY w.$ reads up to the largest multiple of 8 that is less than $w$ before converting the data.

**Default** 8  

**Range** 1–32767

### Details

The $BINARY w.$ informat does not interpret actual binary data, but it converts a string of characters that contains only 0s or 1s as if it is actual binary information. Therefore, use only the character digits 1 and 0 in the input, with no embedded blanks. $BINARY w.$ ignores leading and trailing blanks.

To read representations of binary codes for unprintable characters, enter an ASCII or EBCDIC equivalent for a particular character as a string of 0s and 1s. The $BINARY w.$ informat converts the string to its equivalent character value.

### Comparisons

- The $BINARY w.$ informat reads eight characters of input that contain only 0s or 1s as a binary representation of 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>----------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>0100110001001101</td>
<td>EBCDIC</td>
<td>ASCII</td>
</tr>
</tbody>
</table>

$CB Informat

Reads standard character data from column-binary files.

Category: Column Binary

Syntax

$CBw.

Syntax Description

w specifies the width of the input field.

Default none

Range 1–32767

Details

Column-binary data storage compresses data so that more than 80 items of data can be stored on a single “virtual” punch card.

The $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

    input @1 name $cb2.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100110001001101</td>
<td>EBCDIC</td>
<td>ASCII</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.
$CHAR Informat
Reads character data with blanks.

Category: Character

Syntax
$\text{CHAR}w$.

Syntax Description

\textit{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 $\text{CHAR}w$. informat does not trim leading and trailing blanks or convert a single period in the input data field to a blank before storing values. If you use $\text{CHAR}w$. in an INFORMAT or ATTRIB statement within a DATA step to read list input, then by default SAS interprets any blank embedded within data as a field delimiter, including leading blanks.

Comparisons

- The $\text{CHAR}w$. informat is almost identical to the $w$. informat. However, $\text{CHAR}w$. does not trim leading blanks or convert a single period in the input data field to a blank, while the $w$. informat does.

- Use the table below to compare the SAS informat $\text{CHAR}8$. with notation in other programming languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>Character Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>$\text{CHAR}8$.</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>CL8</td>
</tr>
<tr>
<td>C</td>
<td>char [8]</td>
</tr>
</tbody>
</table>
### Language | Character Notation
---|---
COBOL | PIC x(8)
Fortran | A8
PL/I | CHAR(8)

#### Example

```plaintext
input @1 name $char5.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-----------</td>
<td>--------</td>
</tr>
<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.

---

#### $CHARZB Informat

Converts binary 0s to blanks.

**Category:** Character

#### Syntax

```
$CHARZBw.
```

#### Syntax Description

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

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

- **Range** 1–32767
Details
The $CHARZBw. informat does not trim leading and trailing blanks in character data before it stores values.

Comparisons
The $CHARZBw. informat is identical to the $CHARw. informat except that $CHARZBw. converts any byte that contains a binary 0 to a blank character.

Example

    input @1 name $charzb5.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<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

$EBCDICw.

Syntax Description

w
specifies the width of the input field.

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

Range 1–32767

Details
If EBCDIC is the native format, no conversion occurs.
Note: Anytime a text file originates from anywhere other than the local encoding environment, it might be necessary to specify the ENCODING= option on either ASCII or EBCDIC environments. 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 (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, $EBCDICw$ behaves like the $CHARw$. informat.
• On all other systems, $EBCDICw$. converts EBCDIC data to ASCII.

Example

input @1 name $ebcdic3.

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result *</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
<td>EBCDIC</td>
</tr>
<tr>
<td>qrs</td>
<td>717273</td>
</tr>
<tr>
<td>QRS</td>
<td>515253</td>
</tr>
<tr>
<td>+;&gt;</td>
<td>2B3B3E</td>
</tr>
</tbody>
</table>

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

$HEX Informat

Converts hexadecimal data to character data.

Category: Character

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.
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. SAS interprets values that are read with this informat as ASCII values.

Comparisons
The HEXw. informat reads two digits of hexadecimal data at a time and converts them into one byte of numeric data.

Example
```plaintext
input @1 name $hex4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>6C6C</td>
<td>11</td>
</tr>
</tbody>
</table>

$N8601B 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>yyyy-mm-ddThh:mm:ss.fff</td>
<td>P2012-09-15T15:53:00</td>
</tr>
<tr>
<td></td>
<td>yyyyymmdTthhmmss</td>
<td>P00020304T050607</td>
</tr>
<tr>
<td></td>
<td>PnYnMnDTnHnMn.fffS</td>
<td>P2y10m14dT20h13m45.222s</td>
</tr>
<tr>
<td>Interval</td>
<td>yyyy-mm-ddTthh:mm:ss.fff</td>
<td>2012-09-15T15:53:00/2014-1-13T00:00:00</td>
</tr>
<tr>
<td></td>
<td>yyyyymmdTthhmmss.fff/yyyymmdTthhmmss.fff</td>
<td>20120915T155300/20141115T120000</td>
</tr>
<tr>
<td></td>
<td>PnYnMnDTnHnMn.fffS/ yyyy-mm-ddTthh:mm:ss.fff</td>
<td>P2y10M14dT20h13m45s/2012-09-15T15:53:00</td>
</tr>
<tr>
<td>Datetime</td>
<td>yyyy-mm-ddTthh:mm:ss.fff</td>
<td>2012-09-15T15:53:00</td>
</tr>
<tr>
<td></td>
<td>yyyyymmdTthhmmss.fff</td>
<td>20120915T155300</td>
</tr>
</tbody>
</table>

The $\$N8601B$ informat also reads ISO 8601 duration, interval, and datetime components that contain omitted or truncated components. Omitted components must use a single hyphen ( - ) to represent the component.

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**

```sas
input @1 i860 $n8601b.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0002-04-05t5:1:12</td>
<td>0002405050112FFC</td>
</tr>
<tr>
<td>2012-09-15T15:53:00/2010-09-15T00:00:00</td>
<td>201291515300FFD2010915000000FFD</td>
</tr>
<tr>
<td>p0033-01-04T3:2:55/2012-09-15T15:53:00</td>
<td>0033104030255FFC201291515300FFD</td>
</tr>
</tbody>
</table>

**See Also**

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 243

---

### $N8601E 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.
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><strong>Duration</strong></td>
<td>Pyyyy-mm-ddThh:mm:ss.fff</td>
<td>P2012-09-15T15:53:00</td>
</tr>
<tr>
<td></td>
<td>PnW</td>
<td>P6w</td>
</tr>
<tr>
<td><strong>Interval</strong></td>
<td>yyyy-mm-ddThh:mm:ss.fff/ yyyy-mm-ddThh:mm:ss.fff</td>
<td>2012-09-15T15:53:00/2014-1-13T00:00:00</td>
</tr>
<tr>
<td></td>
<td>yyyy-mm-ddThh:mm:ss.fff/PnYnMnDTnHnMns.fffS</td>
<td>2012-09-15T15:53:00/ P2Y10M14DT20H13M45S</td>
</tr>
<tr>
<td><strong>Datetime</strong></td>
<td>yyyy-mm-ddThh:mm:ss.fff</td>
<td>2012-09-15T15:53:00</td>
</tr>
</tbody>
</table>

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

**P** is the character that is used to indicate that the duration that follows is specified by the number of years, months, days, hours, minutes, and seconds.

**W** is the character that is used to designate that the duration is specified in weeks.

**T** is the character that is used to designate that a time value follows. If all time values are 0, T is not required.

**/** in an interval, is used to separate the beginning and ending datetime values.

**yyyy** specifies a four-digit year.

**mm** specifies a two-digit month between 01 and 12.

**dd** specifies a two-digit day between 01 and 31.

**hh** specifies a two-digit hour between 00 and 23.

**mm** specifies a two-digit minute between 00 and 59.

**ss** specifies a two-digit second between 00 and 59.
Specifies an optional fraction of a second with a precision of up to three digits, where each digit is between 0 and 9.

Y is the character that is used to designate years in a duration.

M is the character that is used to designate months in a duration.

D is the character that is used to designate days in a duration.

H is the character that is used to designate hours in a duration.

M is the character that is used to designate minutes in a duration.

S is the character that is used to designate seconds in a duration.

Comparisons
The $N8601E informat reads valid durations, intervals, and datetimes that are specified only in the extended notation.

The $N8601B informat reads valid durations, intervals, and datetimes that are specified in either the basic or extended notation.

Use the $N8601E informat when you need to ensure compliance with the extended notation.

Example
input @1 i860 $n8601e.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0002-04-05t1:12s</td>
<td>0002405050112FFC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-09-15T15:53:00/2014-09-15T00:00:00</td>
<td>2012915155300FFD2014915000000FFD</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0033-01-04T3:2:55/2012-09-15T15:53:00</td>
<td>0033104030255FFC2012915155300FFD</td>
</tr>
</tbody>
</table>

See Also
“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 243

$OCTAL Informat
Converts octal data to character data.

Category: Character
Syntax

SOCTAL_\(w\).

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 \(\text{SOCTAL}_w\) 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. \(\text{SOCTAL}_w\) treats every three digits of octal data as one digit of character data, ignoring the extra bit.

Use \(\text{SOCTAL}_w\) to read octal representations of binary codes for unprintable characters. Enter an ASCII or EBCDIC equivalent for a particular character in octal notation. Then use \(\text{SOCTAL}_w\) to convert it to its equivalent character value.

Use only the digits 0 through 7 in the input, with no embedded blanks. \(\text{SOCTAL}_w\) ignores leading and trailing blanks.

Comparisons

The \(\text{OCTAL}_w\) 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>114</td>
<td>&lt;</td>
</tr>
</tbody>
</table>

$PHEX Informat

Converts packed hexadecimal data to character data.

Category: Character

Syntax

SPHEX_\(w\).
Syntax Description

$\text{w}$ specifies the number of bytes in the input.

When you use $\text{SPHEX}_w$. to read packed hexadecimal data, the length of the variable is the number of bytes that are required to store the resulting character value, not $w$. In general, a character variable whose length is implicitly defined with $\text{SPHEX}_w$. has a length of $2^w - 1$.

Default: 2

Range: 1–32767

Details

Packed hexadecimal data are like packed decimal data, except that all hexadecimal characters are valid. In packed hexadecimal data, the value of the low-order nibble has no meaning. In packed decimal data, the value of the low-order nibble indicates the sign of the numeric value that the data represent. The $\text{SPHEX}_w$. informat returns a character value and treats the value of the sign nibble as if it were $X'F'$, regardless of its actual value.

Comparisons

The $\text{PD}_w.d$. informat reads packed decimal data and converts them to numeric data.

Example

input @1 devaddr $\text{sphex2.}$;

<table>
<thead>
<tr>
<th>Data Line*</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00011110000001111</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.

$\text{QUOTE Informat}$

Removes matching quotation marks from character data.

Category: Character

Syntax

$\text{QUOTE}_w$. 

Syntax Description

$\text{w}$ specifies the width of the input field.
$UPCASE Informat

Converts character data to uppercase.

**Category:** Character

## Syntax

\[ $UPCASE \]

## Syntax Description

\[ w \]

- \( w \) 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.;
```
$VARYING Informat

Reads character data of varying length.

Category: Character

Syntax

$VARYING\, w, \, length-variable

Syntax Description

$w$

specifies the maximum width of a character field for all the records in an input file.

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

Range 1–32767

$length-variable$

specifies a numeric variable that contains the width of the character field in the current record. SAS obtains the value of $length-variable$ by reading it directly from a field that is described in an INPUT statement or by calculating its value in the DATA step.

Restriction

$length-variable$ cannot be an array reference.

Requirement

You must specify $length-variable$ immediately after $VARYING\, w, \, length-variable$ 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 \textit{length-variable} is greater than or equal to \( w \), SAS reads \( w \) columns.

**Details**

Use \$\texttt{VARYING}w\$. when the length of a character value differs from record to record. After reading a data value with \$\texttt{VARYING}w\$, the pointer's position is set to the first column after the value.

**Examples**

**Example 1: Obtaining a Current Record Length Directly**

```sas
input fwidth 1. name $varying9. fwidth;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>5shark</td>
<td>shark</td>
</tr>
<tr>
<td>3sunfish</td>
<td>sun</td>
</tr>
<tr>
<td>@bluefish</td>
<td>bluefish</td>
</tr>
</tbody>
</table>

* Notice the result of reading the second data line.

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

Use the \texttt{LENGTH=} option in the \texttt{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 \texttt{LENGTH=} option in the \texttt{INFILE} statement assigns the internally stored record length to \texttt{RECLEN} when the first \texttt{INPUT} statement executes. The trailing @ holds the record for another \texttt{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 \texttt{FWIDTH} contains the length of the last field and becomes the \textit{length-variable} argument to the \$\texttt{VARYING}20. informat.

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>----------</td>
<td>--------</td>
</tr>
<tr>
<td>PATEL CHEMISTRY</td>
<td>PATEL CHEMISTRY</td>
</tr>
</tbody>
</table>
$w$ Informat
Reads standard character data.

Category: Character

Alias: $FWw$.

Syntax

$w$.

Syntax Description

$w$

specifies the width of the input field. You must specify $w$ because SAS does not supply a default value.

Range 1–32767

Details

The $w$. informat trims leading blanks and left aligns the values before storing the text. In addition, if a field contains only blanks and a single period, $w$. converts the period to a blank because it interprets the period as a missing value. The $w$. informat treats two or more periods in a field as character data.

Comparisons

The $w$. informat is almost identical to the $CHARw$. informat. However, $CHARw$. does not trim leading blanks nor does it convert a single period in an input field to a blank, 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>
</tbody>
</table>
ANYDTDTE Informat

Reads and extracts the date value from various date, time, and datetime forms.

Category: Date and Time

Syntax

\[ \text{ANYDTDTE}w. \]

Syntax Description

\(w\)

specifies the width of the input field.

Default 9

Range 5–32

Details

The ANYDTDTE informat reads input data that corresponds to any of the following formats 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>DDMYY</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>
</tbody>
</table>

* The character # represents a blank space.
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

```sas
input dateinfo anydtdte21.;
```

<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>01012012</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>
</tbody>
</table>
### ANYDTDTM Informat

**Reads and extracts datetime values from various date, time, and datetime forms.**

<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>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**

**Informs:**

- “ANYDTDTM Informat” on page 281
- “ANYDTTME Informat” on page 285
- “DATE Informat” on page 293
- “DATETIME Informat” on page 295
- “DDMMYY Informat” on page 296
- “JULIAN Informat” on page 315
- “MDYAMPM Informat” on page 316
- “MMDDYY Informat” on page 318
- “MONYY Informat” on page 319
- “TIME Informat” on page 354
- “YMDDTTM Informat” on page 400
- “YYMMDD Informat” on page 362
- “YYQ Informat” on page 365

**Syntax**

```
ANYDTDTM
```
**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>MMDD&lt;YY&gt;YY ** ***</td>
<td>010112</td>
</tr>
<tr>
<td></td>
<td>01012012</td>
</tr>
<tr>
<td>MM&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>
</tbody>
</table>
Informat or Form of Input | Example Data
--- | ---
<YY>YY:MM * ** | 12/01
 | 2012/01

month-day-year | January 1, 2012

month-day-year hh:mm:ss | January 1, 2012 12:00:00

* x is a special character that separates the month from the year.
* ** <YY> indicates the century is optional.
*** IF the month and day values are ambiguous, the value for the DATESTYLE= system option is used to determine the order.

If the input value is a time-only value, then SAS assumes a date of 01JAN1960. If the input value is a date-only value, then SAS assumes a time of 12:00 midnight. Input time values must include hours and minutes. If any part of a date in the input value is missing in the input value, or if the hour and minutes in a time value are missing or out of range, then the value read is a SAS missing value.

The input values for the preceding informats are mutually exclusive except for MMDDYY, DDMMYY, or YYMMDD when two-digit years are used. It is possible for input data such as 01-02-03 or 01-02 to be ambiguous with respect to the month, day, and year. In this case, the DATESTYLE system option indicates the order of the month, day, and year.

The ANYDTTME informat uses the following rules when reading colons and periods in time values:

| Use of Colons and Periods | Example |
--- | ---
a single colon in the value h:m indicates hours and minutes | 14:30 |
two colons in the value h:m:s indicate hours, minutes, and seconds | 14:30:08 |
a single period in the value m:s.ff, where ff is a fraction of a second, indicates that the number preceding the period is the number of seconds | 2:39.66 |
multiple periods in the value indicate that the period is a delimiter for dates and the value is not a time value. | 12.25.2012 |

Comparisons

The ANYDTDTE informat extracts the date part from the derived value. The ANYDTDTM informat extracts the datetime part. The ANYDTTME informat extracts the time part.

Example

```plaintext
input dateinfo anydtdtm21.;
```
### Data Line

<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>DDMMYY</td>
<td>1640995200</td>
<td>01JAN12:00:00:00</td>
</tr>
<tr>
<td>20120101</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>01JAN12:00:00:00</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>
<tr>
<td>January 1, 2012</td>
<td>month-day-year</td>
<td>1546387200</td>
<td>01JAN12:00:00:00</td>
</tr>
</tbody>
</table>

### See Also

**Informats:**
- “ANYDTDTE Informat” on page 279
- “ANYDTTME Informat” on page 285
- “DATE Informat” on page 293
- “DATETIME Informat” on page 295
- “DDMMYY Informat” on page 296
- “JULIAN Informat” on page 315
- “MMDDYY Informat” on page 318
- “MONYY Informat” on page 319
- “TIME Informat” on page 354
- “YYMMDD Informat” on page 362
- “YYQ Informat” on page 365
ANYDTTME Informat

Reads and extracts time values from various date, time, and datetime forms.

**Category:** Date and Time

**Syntax**

\texttt{ANYDTTME} \texttt{w}.

**Syntax Description**

\texttt{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.

<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>JAN2012</td>
</tr>
<tr>
<td>DATETIME</td>
<td>01JAN12 14:30:08</td>
<td>YYMMDD</td>
<td>120101</td>
</tr>
<tr>
<td></td>
<td>01JAN2012 14:30:08.5</td>
<td></td>
<td>20120101</td>
</tr>
<tr>
<td>DDMMYY</td>
<td>010112</td>
<td>YYQ</td>
<td>12Q1</td>
</tr>
<tr>
<td></td>
<td>01012012</td>
<td></td>
<td>2012Q1</td>
</tr>
<tr>
<td>JULIAN</td>
<td>12001</td>
<td>YYQ</td>
<td>12Q1</td>
</tr>
<tr>
<td></td>
<td>2012001</td>
<td></td>
<td>2012Q1</td>
</tr>
<tr>
<td>MMDDYY</td>
<td>010112</td>
<td>\textit{month-day-year}</td>
<td>January 1, 2012</td>
</tr>
<tr>
<td></td>
<td>01012012</td>
<td></td>
<td>2012-01</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 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 $h:m$ indicates hours and minutes</td>
<td>14:30</td>
</tr>
<tr>
<td>two colons in the value $h:m:s$ indicate hours, minutes, and seconds</td>
<td>14:30:08</td>
</tr>
<tr>
<td>a single period in the value $m:s.ff$, where $ff$ is a fraction of a second, indicates that the number preceding the period is the number of seconds</td>
<td>2:39.66</td>
</tr>
<tr>
<td>multiple periods in the value indicate that the period is a delimiter for dates and the value is not a time value.</td>
<td>12.25.2012</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**

```
    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 14:30:08.5</td>
<td>DATETIME</td>
<td>52208.5</td>
<td>14:30:09</td>
</tr>
<tr>
<td>010112</td>
<td>DDMMYY</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>
</tbody>
</table>
BINARY Informat

Converts positive binary values to integers.

**Category:** Numeric

**Syntax**

BINARY<sub>w,d</sub>

**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.

**See Also**

- “ANYTDTE Informat” on page 279
- “ANYDTD TM Informat” on page 281
- “DATE Informat” on page 293
- “DATE TIME Informat” on page 295
- “DDMMYY Informat” on page 296
- “JULIAN Informat” on page 315
- “MMDDYY Informat” on page 318
- “MONYY Informat” on page 319
- “TIME Informat” on page 354
- “YYMMDD Informat” on page 362
- “YYQ Informat” on page 365
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

\[
\text{input @1 value binary8.1;}
\]

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001111</td>
<td>1.5</td>
</tr>
</tbody>
</table>

**BITS 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**

BIT\(S_{w,d}\)

**Syntax Description**

\(w\)

specifies the number of bits to read.

Default 1

Range 1–64

\(d\)

specifies the zero-based offset.

Range 0–63

**Details**

The BIT\(S_{w,d}\) informat extracts particular bits from an input stream and assigns the numeric equivalent of the extracted bit string to a variable. Together, the \(w\) and \(d\) values specify the location of the string that you want to read.
This informat is useful for extracting data from system records with many pieces of information packed into single bytes.

**Example**

```
input @1 value bits4.1;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>----+----1----+</td>
<td>8</td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
</tbody>
</table>

* The EBCDIC binary code for a capital B is 11000010, and the ASCII binary code is 01000010.

The input pointer moves to column 2 (\(d=1\)). Then the INPUT statement reads four bits (\(w=4\)), which is the bit string 1000, and stores the numeric value 8, which is equivalent to this binary combination.

---

**BZ Informat**

Converts blanks to 0s.

**Category:** Numeric

**Syntax**

\[BZ_{w,d}\]

**Syntax Description**

\(w\)

- Specifies the width of the input field.
  - Default: 1
  - Range: 1–32

\(d\)

- Specifies the power of 10 by which to divide the value. If the data contain decimal points, the \(d\) value is ignored. This argument is optional.
  - Range: 0–31

**Details**

The \(BZ_{w,d}\) informat reads numeric values, converts any trailing or embedded blanks to 0s, and ignores leading blanks.

The \(BZ_{w,d}\) informat can read numeric values that are located anywhere in the field. Blanks can precede or follow the numeric value, and a minus sign must precede negative values. The \(BZ_{w,d}\) informat ignores blanks between a minus sign and a numeric value in an input field.
The BZ\textsubscript{w.d} informat interprets a single period in a field as a 0. The informat interprets multiple periods or other nonnumeric characters in a field as a missing value.

To use BZ\textsubscript{w.d} in a DATA step with list input, change the delimiter for list input with the DLM= or DLMSTR= option in the INFILE statement. By default, SAS interprets blanks between values in the data line as delimiters rather than 0s.

**Comparisons**

The BZ\textsubscript{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 \textit{w.d} informat or the E\textit{w.d} informat instead.

**Example**

```plaintext
input @1 x bz4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>34</td>
<td>3400</td>
</tr>
<tr>
<td>-2</td>
<td>-200</td>
</tr>
<tr>
<td>-2 1</td>
<td>-201</td>
</tr>
</tbody>
</table>

**CB Informat**

Reads standard numeric values from column-binary files.

**Category:** Column Binary

**Syntax**

\texttt{CBw.d}

**Syntax Description**

\texttt{w}

specifies the width of the input field.

Range: 1–32

\texttt{d}

specifies the power of 10 by which to divide the value. SAS uses the \textit{d} value even if the data contain decimal points. This argument is optional.

**Details**

Column-binary data storage compresses data so that more than 80 items of data can be stored on a single “virtual” punch card.
The CB\textsubscript{w,d} informat reads standard numeric values from column-binary files and translates the data into standard binary format.

SAS first stores each column of column-binary data that you read with CB\textsubscript{w,d} in two bytes and ignores the two high-order bits of each byte. If the punch codes are valid, then SAS stores the equivalent numeric value in the variable that you specify. If the combinations are not valid, then SAS assigns the variable a missing value and sets the automatic variable \_ERROR\_ to 1.

**Example: Examples**

```
input @1 x cb8.;
```

<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**

- “$CB Informat” on page 263
- “PUNCH Informat” on page 331
- “ROW Informat” on page 392

---

**COMMA Informat**

Removes embedded characters.

**Category:** Numeric  
**Alias:** DOLLAR\textsubscript{w,d}

**Syntax**

COMMA\textsubscript{w,d}

**Syntax Description**

\texttt{w} specifies the width of the input field.

<table>
<thead>
<tr>
<th>Default</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1–32</td>
</tr>
</tbody>
</table>
\( d \)
specifies the power of 10 by which to divide the value. If the data contain decimal points, the \( d \) value is ignored. This argument is optional.

Range 0–31

Details
The COMMA\( w.d \) informat reads numeric values and removes embedded commas, blanks, dollar signs, percent signs, hyphens, and close parentheses from the input data. The COMMA\( w.d \) informat converts an open parenthesis at the beginning of a field to a minus sign.

Comparisons
The COMMA\( w.d \) informat operates like the COMMAX\( w.d \) informat, but it reverses the roles of the decimal point and the comma. This convention is common in European countries.

Example
\[
\begin{align*}
\text{input @1 x comma10.;} \\
\text{Data Line} & \quad \text{Result} \\
\hline
\$1,000,000 & \quad 1000000 \\
(500) & \quad -500 \\
\hline
\end{align*}
\]

COMMAX Informat
Remove 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.

\begin{itemize}
  \item **Category:** Numeric
  \item **Alias:** DOLLARX\( w.d \)
\end{itemize}

Syntax
COMMAX\( w.d \)

Syntax Description
\( w \)

specifies the width of the input field.

\begin{itemize}
  \item **Default:** 1
\end{itemize}
Range 1−32

d specifies the power of 10 by which to divide the value. If the data contain a comma, which represents a decimal point, the d value is ignored. This argument is optional.

Range 0−31

Details
The 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>

DATE Informat
Reads date values in the form ddmmyy or ddmmyyyy.

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 date11.;
```

<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:
- “DATE Format” on page 141

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

System Options:
- “YEARCUTOFF= System Option” in *SAS Viya System Options: Reference*
DATETIME Informat

Reads datetime values in the form `ddmmmyy hh:mm:ss.ss` or `ddmmmyyyy hh:mm:ss.ss`.

**Category:** Date and Time

**Syntax**

`DATETIMEw.`

**Syntax Description**

`w`

- specifies the width of the input field.

  **Default** 18
  
  **Range** 13–40

**Details**

The datetime values must be in the following form: `ddmmmyy` or `ddmmmyyyy`, followed by a blank or special character, followed by `hh:mm:ss.ss` (the time):

- `dd` is an integer between 01 and 31 that represents the day of the month.
- `mmm` is the first three letters of the month name.
- `yy` or `yyyy` is a two-digit or four-digit integer that represents the year.
- `hh` is an integer between 00 and 23 that represents hours.
- `mm` is an integer between 00 and 59 that represents minutes.
- `ss.ss` is the number of seconds ranging from 00–59 with the fraction of a second following the decimal point.

`DATETIMEw.` requires values for both the date and the time. However, the `ss.ss` portion is optional.

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

**Note:** SAS can read time values with AM and PM in them.

**Comparisons**

The `DATETIMEw.d` informat reads datetime values with optional separators in the form `dd-mm-yy<yy> hh:mm:ss AM|PM`, and the date and time can be separated by a special character.
The MDYAMPMw.d in format reads datetime values with optional separators in the form \textit{mm-dd-yy<yy>} hh:mm:ss.ss AM | PM, and requires a space between the date and the time.

The YMDDTTMw.d informat reads datetime values with required separators in the form \textit{<yy>yy-mm-dd/hh:mm:ss.ss}.

**Example**

```sas
   input date_and_time datetime20.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------------------------------</td>
<td>----------</td>
</tr>
<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**

- “DATE Format” on page 141
- “DATETIME Format” on page 56
- “TIME Format” on page 98

**Formats:**

- “DATE Format” on page 141
- “DATETIME Format” on page 56
- “TIME Format” on page 98

**Functions:**

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

**Informat:**

- “DATE Informat” on page 293
- “MDYAMPM Informat” on page 316
- “TIME Informat” on page 354
- “YMDDTTM Informat” on page 400

**System Options:**

- “YEARCUTOFF= System Option” in \textit{SAS Viya System Options: Reference}

---

**DDMMYY Informat**

Reads date values in the form \textit{ddmmyy<yy>} or \textit{dd-mm-yy<yy>}, where a special character, such as a hyphen (-), period (.), or slash (/), separates the day, month, and year; the year can be either 2 or 4 digits.

**Category:** Date and Time
Syntax

```
DDMMYY\^w.
```

**Syntax Description**

\^w

specifies the width of the input field.

- **Default**: 6
- **Range**: 6–32

**Details**

The date values must be in the form `ddmmyy\<yy>` or `ddxmmxyy\<yy>`:

- `dd` is an integer between 01 and 31 that represents the day of the month.
- `mm` is an integer between 01 and 12 that represents the month.
- `yy` or `yyyy` is a two-digit or four-digit integer that represents the year.
- `x` is a separators that can be any special character or a blank.

If you use separators, place them between all the values. Blanks can also be placed before and after the date. Make sure the width of the input field allows space for blanks and special characters.

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

**Example**

```
input calendar_date dmmmyy10.;
```

<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:
- “DATE Format” on page 141
- “DDMYY Format” on page 58
- “MMDDYY Format” on page 82
- “YYMMDD Format” on page 117

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

Informats:
- “DATE Informat” on page 293
- “MMDDYY Informat” on page 318
- “YYMMDD Informat” on page 362

System Options:
- “YEARCUTOFF= System Option” in SAS Viya System Options: Reference

E8601DA Informat

Reads date values that are specified using the ISO 8601 extended notation yyyy-mm-dd.

**Categories:** Date and Time

**Alignment:** Left

**Alias:** IS8601DAw

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

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

**Syntax**

E8601DA\[w\].

**Syntax Description**

\[w\]

specifies the width of the input field.

**Default** 10

**Requirement** The width of the input field must be 10.
Details

The E8601DA informat reads date values that are specified in the ISO 8601 extended date notation yyyy-mm-dd:

- **yyyy**
  - is a four-digit year.
- **mm**
  - is a two-digit month (zero padded) between 01 and 12.
- **dd**
  - is a two-digit day of the month (zero padded) between 01 and 31.

Example

```plaintext
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 243

---

**E8601DN 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**

```plaintext
E8601DNw.
```

**Syntax Description**

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

  **Default**

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

Details
The E8601DN informat reads date values that are specified using the ISO 8601 extended date notation \texttt{yyyy-mm-dd} and returns the date in a SAS datetime value:

\begin{itemize}
  \item \texttt{yyyy} is a four-digit year.
  \item \texttt{mm} is a two-digit month (zero padded) between 01 and 12.
  \item \texttt{dd} is a two-digit day of the month (zero padded) between 01 and 31.
\end{itemize}

Example

\begin{verbatim}
input edn e8601dn.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-09-15</td>
<td>1663286400</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 243

E8601DT Informat
Reads datetime values that are specified using the ISO 8601 extended notation \texttt{yyyy-mm-ddThh:mm:ss.<ffffff>},

- **Categories:** Date and Time
  - ISO 8601
- **Alignment:** Left
- **Aliases:** E8601LX\(w.d\)
  - IS8601DT\(w.d\)
- **Restriction:** UTC time zone offset values are not supported.
- **Supports:** ISO 8601 Element 5.4.1, complete representation

Syntax

\begin{verbatim}
E8601DT\(w.d\)
\end{verbatim}
Syntax Description

\( w \)

specifies the width of the input field.

Default 19
Range 19–26

\( d \)

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

Default 0
Range 0–6

Details

The E8601DT informat reads datetime values that are specified using the ISO 8601 extended datetime notation \( yyyy-mm-ddThh:mm:ss.ffffff <ffffff> \):

\( yyyy \)

is a four-digit year.

\( mm \)

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

\( dd \)

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

\( hh \)

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

\( mm \)

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

\( ss \)

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

\( fffffff \)

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

Example

input @1 edt e8601dt.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-09-15T15:53:00</td>
<td>1663343580</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 243
E8601DX 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 yyyy-mm-ddThh:mm:ss±|–hh:mm.

See: The E8601DXw informat uses the E8601DZw.d informat to read data. See “E8601DZ Informat” on page 302.

Syntax

E8601DXw.

E8601DZ 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.<fffff> or yyyy-mm-ddThh:mm:ss.<fffff>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:

- \texttt{yyyy-mm-ddThh:mm:ss.<ffffff>\pm.hh:mm}
- \texttt{yyyy-mm-ddThh:mm:ss.<ffffff>Z}

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

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

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

\textbf{Restriction}: The shorter form \texttt{\pm hh} is not supported.

\texttt{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>input edz e8601dz.;</td>
<td>2012-09-15T15:53:00Z</td>
<td>1663343580</td>
</tr>
<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 243
E8601TM Informat

Reads time values that are specified using the ISO 8601 extended notation \( hh:mm:ss.<ffffffff> \).

**Categories:** Date and Time
ISO 8601

**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**

\[ \text{E8601TM}w.d \]

**Syntax Description**

\( w \)

specifies the width of the input field.

Default 8

Range 8–15

\( d \)

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

Default 0

Range 0–6

**Details**

The E8601TM informat reads time values that are specified using the ISO 8601 extended time notation \( hh:mm:ss.<ffffffff> \):

\( hh \)

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

\( mm \)

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

\( ss \)

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

\( <ffffffff> \)

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

**Example**
input @1 etm e8601tm.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>15:53:00</td>
<td>57180</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 243

E8601TZ Informat

Reads time values that are specified using the ISO 8601 extended time notation `hh:mm:ss+|-hh:mm.<fff>`, or `hh:mm:ssZ`.

- **Categories:** Date and Time, ISO 8601
- **Alignment:** Left
- **Aliases:** E8601TXw.d, IS8601TZw.d
- **Supports:** ISO 8601 Element 5.3.1.1, complete representation

**Syntax**

`E8601TZw.d`

**Syntax Description**

- `w` specifies the width of the input field.
  - *Default:* 14
  - *Range:* 9–20
  - *Requirement:* To read a time with the Z time zone indicator, the width of the input field must be 9 if data follows on the same line of data.

- `d` specifies the number of digits to the right of the decimal point in the value for the lowest-order component. This argument is optional.
  - *Default:* 0
  - *Range:* 0–6
Details
UTC time values specify a time and a time zone based on the zero meridian in Greenwich, England. The E8601TZ informat reads UTC time values that are specified in one of the following ISO 8601 extended notations:

- \texttt{hh:mm:ss+|--hh:mm.<ffffff>}
- \texttt{hh:mm:ssZ}

\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.

\texttt{+|--hh:mm}

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

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

\texttt{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

\begin{verbatim}
input @1 etz e8601tz.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>23:43:44-05:00</td>
<td>17024</td>
</tr>
<tr>
<td>23:43:44Z</td>
<td>85424</td>
</tr>
</tbody>
</table>

See Also

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 243
FLOAT Informat

Reads a native single-precision, 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

```
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 FLOAT\(w.d\) informat is useful in operating environments where a float value is not the same as a truncated double.

On the IBM mainframe systems, a four-byte floating-point number is the same as a truncated eight-byte floating-point number. However, in operating environments that use the IEEE floating-point standard, such as the IBM PC-based operating environments and LINUX, 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

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

### 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.

### HEX Informat

Converts hexadecimal positive binary values to either integer (fixed-point) or real (floating-point) binary values.

**Category:** Numeric

**Syntax**

`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.

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.

**Default** 8

**Range** 1–16

**Details**

The `HEXw` informat converts the hexadecimal representation of positive binary numbers to real floating-point binary values. The width value of the `HEXw` informat determines whether the input represents an integer (fixed-point) or real (floating-point) binary number. When you specify a width of 1 through 15, the informat interprets the input hexadecimal as an integer binary number. When you specify 16 for the width value, the informat interprets the input hexadecimal as a floating-point value.

The `HEXw` informat ignores leading or trailing blanks.
Example

```plaintext
input @1 x hex3. @5 y hex16. ;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>88F 40148000000000</td>
<td>2191 5.125</td>
</tr>
</tbody>
</table>

**HHMMSS Informat**

Reads hours, minutes, and seconds in the form `hh:mm:ss` or `hhmmss`.

**Category:** Date and Time

**Syntax**

`HHMMSSw`.

**Syntax Description**

`w`

- specifies the width of the input field.

  **Default:** 8

  **Range:** 1–20

**Details**

The `HHMMSSw` informat reads SAS time values in one of the following forms:

- `hh:mm:ss`
- `hhmmss`

- `hh`
  - is an integer that represents the number of hours.

- `:`
  - represents a special character that separates hours, minutes, and seconds.

- `mm`
  - is an integer that represents the number of minutes.

- `ss`
  - is an integer that represents the number of seconds. Fractional seconds are ignored.

If the input data is six digits, 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

```sas
input tm hhmmss;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Formatted with TIMExw.</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:
- “TIME Informat” on page 354
IB 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:**

---

**Syntax**

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

**Syntax Description**

\[ w \]

specifies the width of the input field.

- **Default:** 4
- **Range:** 1–8

\[ d \]

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

- **Range:** 0–10

**Details**

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

*Note:* Different operating environments store integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 237.

**Comparisons**

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

To view a table that shows the type of informat to use with big endian and little endian integers, see “Reading Data Generated on Big Endian or Little Endian Platforms” on page 238.

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:
- “IBR Informat” on page 312

IBR 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**

```sas
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 IB\textit{w}.\textit{d} informat reads integer binary (fixed-point) values, including negative values that are represented in two's complement notation. IB\textit{w}.\textit{d} reads integer binary values that are generated by and for Intel and DEC platforms. Use IB\textit{w}.\textit{d} to read integer binary data from Intel or DEC environments in other operating environments. The IB\textit{w}.\textit{d} informat in SAS code allows for a portable implementation for reading the data in any operating environment.

\textit{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 237.

Comparisons

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 IB\textit{w}.\textit{d} and PIBR\textit{w}.\textit{d} informats are used to read little endian integers in any operating environment.

On Intel and DEC operating environments, the IB\textit{w}.\textit{d} and IBR\textit{w}.\textit{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 238.

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, in these examples that we use the informat with the INPUT function, where binary input values are described using a hexadecimal literal.

\begin{verbatim}
x=input('0100'x,ibr2.);
y=input('0001'x,ibr2.);
\end{verbatim}

<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:

- “IB Informat” on page 311

IEEE Informat

Reads an IEEE floating-point value and divides it by 10 raised to the \textit{d} th power.
Syntax

\texttt{IEEE}w.d

Syntax Description

\texttt{w}

specifies the width of the input field.

\begin{itemize}
\item \textbf{Default} \hspace{1cm} 8
\item \textbf{Range} \hspace{1cm} 2–8
\end{itemize}

\textbf{Tip}

If \texttt{w} is 8, an IEEE double-precision, floating-point number is read. If \texttt{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 \texttt{w} is 4, an IEEE single-precision, floating-point number is read. If \texttt{w} is 3, an IEEE single-precision, floating-point number is read, which assumes truncation of one byte.

\texttt{d}

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

Details

The IEEE\texttt{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\texttt{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\texttt{w.d} informat allows other lengths, which enables you to read data from files that contain space-saving truncated data.

Example

\begin{verbatim}
input test1 ieee4.;
input test2 ieee5.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>3F800000</td>
<td>1</td>
</tr>
<tr>
<td>3FF000000</td>
<td>1</td>
</tr>
</tbody>
</table>

* The data lines are hexadecimal representations of binary numbers that are stored in IEEE format.
The first INPUT statement reads the first data line, and the second INPUT statement reads the next data line.

**JULIAN Informat**

Reads Julian dates in the form *yyddd* or *yyyyddd*.

**Syntax**

```
JULIANw.
```

**Syntax Description**

*w*

specifies the width of the input field.

**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>---------</td>
<td>------</td>
</tr>
<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:
- “JULIAN Format” on page 80

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

System Options:
- “YEARCUTOFF= System Option” in SAS Viya System Options: Reference

MDYAMPM 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 `mm-dd-yy<yy>`
`hh:mm:<ss><.ss><AM | PM>`:

- `mm` is an integer between 01 and 12 that represents the month.
- `dd` is an integer between 01 and 31 that represents the day of the month.
- `yy` or `yyyy` specifies a two-digit or four-digit integer that represents the year.
- `hh` is an integer between 00 and 23 that represents hours.
- `mm` is an integer between 00 and 59 that represents minutes.
- `ss.ss` is the number of seconds that range from 00–59 with the fraction of a second
  following the decimal point.

**Requirement:** If a fraction of a second is specified, the decimal point can be
represented only by a period and is required.

- `AM | PM` specifies either the time period 00:01–12:00 noon (AM) or the time period 12:01–
12:00 midnight (PM)
- `:` 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 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-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
`<yy>yy-mm-dd/hh:mm:ss.ss`.

Example

```
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>
MMDDYY Informat
Reads date values in the form mmddyy or mmddyyyy.

Category: Date and Time

Syntax

**MMDDYY**\(w\).

**Syntax Description**

\(w\)

specifies the width of the input field.

Default 6

Range 6–32

**Details**

The date values must be in one of the following forms:

- \(mmddyy\)
- \(mmddyyyy\)

\(mm\)

is an integer between 01 and 12 that represents the month.

\(dd\)

is an integer between 01 and 31 that represents the day of the month.

\(yy\) or \(yyyy\)

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

You can separate the month, day, and year fields by blanks or by special characters. However, if you use delimiters, place them between all fields in the value. Blanks can also be placed before and after the date.

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

    input calendar_date mmddyy8.;

<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:
- “DATE Format” on page 141
- “DDMMYY Format” on page 58
- “MMDDYY Format” on page 82
- “YYMMDD Format” on page 117

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

Informats:
- “DATE Informat” on page 293
- “DDMMYY Informat” on page 296
- “YYMMDD Informat” on page 362

System Options:
- “YEARCUTOFF= System Option” in SAS Viya System Options: Reference

MONYY Informat

Reads month and year date values in the form mmyy or mmyyyy.

Category: Date and Time
Syntax

\texttt{MONYY}_w.

Syntax Description

\textit{w}

specifies the width of the input field.

\textbf{Default} \hspace{2cm} 5

\textbf{Range} \hspace{2cm} 5–32

Details

The date values must be in one of the following forms:

\begin{itemize}
  \item \texttt{mmmyy}
  \item \texttt{mmmyyyy}
\end{itemize}

\textit{mmm}

is the first three letters of the month name.

\textit{yy} or \textit{yyyy}

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

A value read with the \texttt{MONYY}_w. informat results in a SAS date value that corresponds to the first day of the specified month.

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

Example

\begin{verbatim}
input month_and_year monyy7.;
\end{verbatim}

\begin{tabular}{l|l}
\textbf{Data Line} & \textbf{Result} \\
\hline
----------1 & 19053  \\
mar 12 & 19053  \\
mar2012 & 19053  \\
\end{tabular}

See Also

Formats:

\begin{itemize}
  \item “DDMMYY Format” on page 58
  \item “MMDDYY Format” on page 82
  \item “MONYY Format” on page 92
  \item “YYMMDD Format” on page 117
\end{itemize}
**MSEC 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:
• “TODSTAMP Informat” on page 356

NUMX Informat
Reads numeric values with a comma in place of the decimal point.

Category: Numeric

Syntax

NUMXw.d

Syntax Description

w
specifies the width of the input field.

Default 12
Range 1-32

d
specifies the number of digits to the right of the decimal. If the data 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 numx10.;
Data Line | Result
---+----1----+  
896.48 | 896.48
3064.1 | 3064.1
6489 | 6489

See Also

Formats:
- “NUMX Format” on page 189
- “w Format” on page 105

PD Informat

Reads data that are stored in IBM packed decimal 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

`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, PD\textsubscript{w}d reads packed decimal values with consistent results if the values are created on the same type of operating environment that you use to run SAS.

The PD\textsubscript{w}d format writes missing numerical data as -0. When the PD\textsubscript{w}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**

input @1 x pd4.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>0000128C</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**

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>18621</td>
<td>012252010C</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.
PDTIME Informat

Reads packed decimal time of SMF and RMF records.

**Syntax**

\[
PDTIME_w.
\]

**Syntax Description**

\[w\]

specifies the width of the input field.

**Requirement**

\[w\] must be 4 because packed decimal time values in RMF and SMF records contain four bytes of information.

**Details**

The PDTIME\(_w\). informat reads packed decimal time values that are contained in SMF and RMF records that are produced by IBM mainframe systems and converts the values to SAS time values.

The general form of a packed decimal time value in hexadecimal notation is 0\(_{hhmmss}\)F:

- 0 is a half byte that contains all 0s.
- \(hh\) is one byte that represents two digits that correspond to hours.
- \(mm\) is one byte that represents two digits that correspond to minutes.
- \(ss\) is one byte that represents two digits that correspond to seconds.
- F is a half byte that contains all 1s.

If a field contains all 0s, PDTIME\(_w\). treats it as a missing value.

PDTIME\(_w\). enables you to read packed decimal time values from files that are created on an IBM mainframe on any operating environment.

**Example**

```plaintext
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.*
PERCENT Informat
Reads percentages as numeric values.

Syntax
PERCENT\textsubscript{\textit{w.d}}

Syntax Description
\textit{w}

specifies the width of the input field.

Default 6
Range 1–32

\textit{d}

specifies the power of 10 by which to divide the value. If the data contain decimal points, the \textit{d} value is ignored. This argument is optional.

Range 0–31

Details
The \textit{PERCENTw.d} informat converts the numeric portion of the input data to a number using the same method as the \textit{COMMAw.d} informat. If a percent sign (%) follows the number in the input field, \textit{PERCENTw.d} divides the number by 100.

Example

\texttt{input @1 x \text{percent}3. @4 y \text{percent}5.;}

<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>

PIB 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.

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. PIBw.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 237.

Comparisons

• Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIBw.d informat treats all values as positive and includes the sign bit as part of the value.

• The PIBw.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 IBw.d and PIBw.d informats are used to read native format integers. (Native format enables you to read and write values that are created in the same operating environment.) The IBRw.d and PIBRw.d informats are used to read little endian integers in any operating environment.

To view a table that shows the type of informat to use with big endian and little endian integers, see “Reading Data Generated on Big Endian or Little Endian Platforms” on page 238.

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.

```sas
x=input('0100'x,pib2.);
y=input('0001'x,pib2.);
```

<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:

- “PIBR Informat” on page 328

PIBR 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**

`PIBRw.d`

**Syntax Description**

`w`

specifies the width of the input field.

- Default: 1
- Range: 1–8

`d`

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

- Range: 0–10

**Details**

All values are treated as positive. `PIBRw.d` reads positive integer binary values that have been generated by and for Intel and DEC operating environments. Use `PIBRw.d` to read positive integer binary data from Intel or DEC environments on other operating systems.
environments. The PIBRw.d informat in SAS code allows for a portable implementation for reading the data in any operating environment.

Note: Different operating environments store positive integer binary values in different ways. This concept is called byte ordering. For a detailed discussion about byte ordering, see “Byte Ordering for Integer Binary Data on Big Endian and Little Endian Platforms” on page 237.

Comparisons

- Positive integer binary values are the same as integer binary values except that the sign bit is part of the value, which is always a positive integer. The PIBRw.d informat treats all values as positive and includes the sign bit as part of the value.

- The PIBRw.d informat with a width of 1 results in a value that corresponds to the binary equivalent of the contents of a byte. This is useful if your data 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 IBw.d and PIBw.d informats are equivalent.

- The IBw.d and PIBw.d informats are used to read native format integers. (Native format enables you to read and write values that are created in the same operating environment.) The IBRw.d and PIBRw.d informats are used to read little endian integers in any operating environment.

To view a table that shows the type of informat to use with big endian and little endian integers, see “Reading Data Generated on Big Endian or Little Endian Platforms” on page 238.

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

Example

You can use the INPUT statement and specify the PIBR informat. However, these examples use the informat with the INPUT function, where binary input values are described using a hexadecimal literal.

    x=input('0100'x,pibr2.);
    y=input('0001'x,pibr2.);

<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 Informat” on page 326
PK 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**

**Syntax Description**

\[ PK_{w.d} \]

- **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

```sas
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 Informat
Reads whether a row of column-binary data is punched.

**Category:** Column Binary

**Syntax**

\[ \text{PUNCH}.d \]

**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**

**Informats:**

- “$CB Informat” on page 263
- “CB Informat” on page 290
- “ROW Informat” on page 392
RB 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.

**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 RB\(_w.d\) informat reads real binary values with consistent results if the values are created on the same type of operating environment that you use to run SAS.

**Comparisons**

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

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>RB4.</td>
<td>RB8.</td>
</tr>
<tr>
<td>Fortran</td>
<td>REAL*4</td>
<td>REAL*8</td>
</tr>
<tr>
<td>C</td>
<td>float</td>
<td>double</td>
</tr>
<tr>
<td>IBM 370 assembler</td>
<td>F</td>
<td>D</td>
</tr>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
</tbody>
</table>
CAUTION: Using the RBw.d informat to read real binary information about equipment that conforms to the IEEE standard for floating-point numbers results in a truncated eight-byte number (double-precision), rather than in a true four-byte floating-point number (single-precision).

Example

```plaintext
input @1 x rb8.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128</td>
</tr>
</tbody>
</table>

* The data line is a hexadecimal representation of a real binary (floating-point) number on an IBM mainframe operating environment. Each byte occupies one column of the input field.

See Also

Informats:
- “IEEE Informat” on page 313

RMFDUR Informat

Reads duration intervals of RMF records.

Category: Date and Time

Syntax

RMFDURw.

Syntax Description

w

specifies the width of the input field.

Requirement w must be 4 because packed decimal duration values in RMF records contain four bytes of information.

Details

The RMFDURw. 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 mmssttttF:
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 RMFDURw. results in a missing value.

**Comparisons**
- Both the RMFDURw. informat and the RMFSTAMPw. informat read packed decimal information from RMF records that are produced by IBM mainframe systems.
- The RMFDURw. 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**

- “RMFSTAMP Informat” on page 334
- “SMFSTAMP Informat” on page 353

**RMFSTAMP Informat**

Reads time and date fields of RMF records.

**Category:** Date and Time
Syntax

RMFSTAMP\(w\).

Syntax Description

\(w\)

specifies the width of the input field.

Requirement \(w\) must be 8 because packed decimal time and date values in RMF records contain eight bytes of information: four bytes of time data that are followed by four bytes of date data.

Details

The RMFSTAMP\(w\). informat reads packed decimal time and date values of RMF records that are produced by IBM mainframe systems, and converts the time and date values to SAS datetime values.

The general form of the time and date information in an RMF record in hexadecimal notation is 0\(hhmmssFccyydddF\):

- 0 is the half byte that contains all binary 0s.
- \(hh\) is the one-byte representation of two digits that correspond to the hour of the day.
- \(mm\) is the one-byte representation of two digits that correspond to minutes.
- \(ss\) is 1 byte that represents two digits that correspond to seconds.
- \(cc\) is the one-byte representation of two digits that correspond to the century.
- \(yy\) is the one-byte representation of two digits that correspond to the year.
- \(ddd\) is the one-and-a-half bytes that contain three digits that correspond to the day of the year.
- F is the half byte that contains all binary 1s.

The century indicators 00 correspond to 1900, 01 to 2000, and 02 to 2100.

RMFSTAMP\(w\). enables you to read, on any operating environment, packed decimal time and date values from files that are created on an IBM mainframe.

Comparisons

Both the RMFSTAMP\(w\). informat and the PDTIME\(w\). informat read packed decimal values from RMF records. The RMFSTAMP\(w\). informat reads both time and date values and results in a SAS datetime value. The PDTIME\(w\). informat reads only time values and results in a SAS time value.

Example
input begin: $hex16.;
y=input(begin, rmfstamp8.);

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>014225F2612200F</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.

---

**S370FF Informat**

Reads EBCDIC numeric data.

**Category:** Numeric

**Syntax**

S370FF<sub>w,d</sub>

**Syntax Description**

**w**

specifies the width of the input field.

- **Default:** 12
- **Range:** 1–32

**d**

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

- **Range:** 0–31

**Details**

The S370FF<sub>w,d</sub> informat reads numeric data that are represented in EBCDIC and converts the data to native format. If EBCDIC is the native format, S370FF<sub>w,d</sub> performs no conversion.

S370FF<sub>w,d</sub> reads EBCDIC numeric values that are represented with one byte per digit. Use S370FF<sub>w,d</sub> on other operating environments to read numeric data from IBM mainframe files.

S370FF<sub>w,d</sub> reads numeric values located anywhere in the input field. EBCDIC blanks can precede or follow a numeric value with no effect. If a value is negative, an EBCDIC minus sign should immediately precede the value. S370FF<sub>w,d</sub> 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

\begin{verbatim}
input @1 x s370ff3.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textit{F1F2F3}</td>
<td>123</td>
</tr>
<tr>
<td>\textit{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.

S370FIB 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

S370FIB\textit{w.d}

\textbf{Syntax Description}

\textit{w}

specifies the width of the input field.

\begin{tabular}{ll}
Default & 4 \\
Range & 1–8 \\
\end{tabular}

\textit{d}

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

\begin{tabular}{ll}
Range & 0–10 \\
\end{tabular}
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 237.

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 238.

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.

```sas
x=input('0080'x,s370fib2.);
```

<table>
<thead>
<tr>
<th>SAS Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>put x=;</td>
<td>128</td>
</tr>
</tbody>
</table>

See Also

Informats

• “S370FIBU Informat” on page 338
• “S370FPIB Informat” on page 342

S370FIBU 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

\texttt{S370FIBU}w.d

\textbf{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 237.

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.
- S370FIPIBw.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 238.

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:
- “S370FIB Informat” on page 337
- “S370FPIB Informat” on page 342

S370FPD 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**

`S370FPDw.d`

**Syntax Description**

`w`

- Specifies the width of the input field.
  - Default: 1
  - Range: 1–16

`d`

- Specifies the power of 10 by which to divide the value. This argument is optional.
  - Default: 0
  - Range: 0–31
**Details**

Packed decimal data contain two digits per byte, but only one digit in the input field represents the sign. The last half of the last byte indicates the sign: a C or an F for positive numbers and a D for negative numbers.

Use S370FPD\textsubscript{w.d} to read packed decimal data from IBM mainframe files on other operating environments.

**Comparisons**

- If you use SAS on an IBM mainframe, the S370FPD\textsubscript{w.d} and the PD\textsubscript{w.d} informats are identical.
- The following table compares the equivalent packed decimal notation by programming language:

<table>
<thead>
<tr>
<th>Language</th>
<th>Packed Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FPD4.</td>
</tr>
<tr>
<td>PL/I</td>
<td>FIXED DEC(7,0)</td>
</tr>
<tr>
<td>COBOL</td>
<td>COMP-3 PIC 9(7)</td>
</tr>
<tr>
<td>assembler</td>
<td>PL4</td>
</tr>
</tbody>
</table>

---

**S370FPDU 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**

S370FPDU\textsubscript{w.d}

**Syntax Description**

\textit{w}

specifies the width of the input field.

- **Default:** 1
- **Range:** 1–16

\textit{d}

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

```
input @1 x s370fpdu3.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345F</td>
<td>12345</td>
</tr>
</tbody>
</table>

* The data line is a hexadecimal representation of a binary number that is stored in packed decimal form. Each two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one column of the input field.

S370FPIB Informat

Reads positive integer binary (fixed-point) values in IBM mainframe 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>

Syntax

S370FPIBw.d

Syntax Description

w

  specifies the width of the input field.
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 237.

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 238.

  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.

```sas
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:
S370FRB 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**

\[ \text{S370FRB} \, w.d \]

**Syntax Description**

\( w \)

specifies the width of the input field.

- **Default:** 6
- **Range:** 2–8

\( d \)

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

- **Range:** 0–10

**Details**

Real binary values are represented in two parts: a mantissa that gives the value, and an exponent that gives the value's magnitude.

Use S370FRB\( w.d \) to read real binary data from IBM mainframe files on other operating environments.

**Comparisons**

- If you use SAS on an IBM mainframe, S370FRB\( w.d \) and RB\( w.d \) are identical.
- The following table shows the equivalent real binary notation for several programming languages:

<table>
<thead>
<tr>
<th>Language</th>
<th>4 Bytes</th>
<th>8 Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>S370FRB4</td>
<td>S370FRB8</td>
</tr>
<tr>
<td>PL/I</td>
<td>FLOAT BIN(21)</td>
<td>FLOAT BIN(53)</td>
</tr>
</tbody>
</table>
### See Also

**Informats:**
- “RB Informat” on page 332

---

**S370FZD 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**

\[ \text{S}370\text{FZD}_{w.d} \]

**Syntax Description**

- \( w \)
  - specifies the width of the input field.
  - **Default:** 8
  - **Range:** 1–32

- \( d \)
  - specifies the power of 10 by which to divide the value. If the data contain decimal points, the \( d \) value is ignored. This argument is optional.
  - **Default:** 0
  - **Range:** 0–31
Details

Zoned decimal data are similar to standard decimal data in that every digit requires one byte. However, the value’s sign is stored in the last byte, along with the last digit.

Use 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

```plaintext
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:

• “ZD Informat” on page 367

S370FZDB 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.

Syntax
S370FZBDw.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 s370fzb8.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>40404040F14040C0</td>
<td>1000</td>
</tr>
<tr>
<td>40404040F1FD3</td>
<td>-123</td>
</tr>
</tbody>
</table>

* The data lines contain a hexadecimal representation of a binary number that is stored in zoned decimal format on an IBM mainframe operating environment. Two hexadecimal characters correspond to one byte of binary data, and each byte corresponds to one column of the input field.

S370FZDL 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

S370FZDLw.d

Syntax Description

w
specifies the width of the input field.

Default 8
Range 1–32

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

Default 0
Range 0–31

Details

Use S370FZDLw.d 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 S370FZDLw.d informat is equivalent to the COBOL notation PIC S9(n) DISPLAY SIGN LEADING, where the n value is the number of digits.

Example

input @1 x 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 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**

\[ \text{S370FZDS}\quad w . d \]

**Syntax Description**

- \( w \)
  - Specifies the width of the input field.
  - Default: 8
  - Range: 2–32

- \( d \)
  - Specifies the power of 10 by which to divide the value. This argument is optional.
  - Default: 0
  - Range: 0–31

**Details**

Use S370FZDS\( w . d \) on other operating environments to read zoned decimal data from IBM mainframe files.

**Comparisons**

- Zoned decimal separate leading-sign data is similar to standard zoned decimal data except that the sign of the value is stored in the first byte of zoned decimal leading sign data, and the first digit of the value is stored in the second byte.
- The S370FZDS\( w . d \) informat is equivalent to the COBOL notation PIC S9(\( n \)) DISPLAY SIGN LEADING SEPARATE, where the \( n \) value is the number of digits.

**Example**

```
    input @1 x s370fzds4. ;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---- ------</td>
<td>------</td>
</tr>
<tr>
<td>1</td>
<td>123</td>
</tr>
<tr>
<td>4EF1F2F3</td>
<td>123</td>
</tr>
</tbody>
</table>
S370FZDT 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**

\[ S370FZDT \]

**Syntax Description**

- \( w \)
  - Specifies the width of the input field.
  - **Default:** 8
  - **Range:** 2–32

- \( d \)
  - Specifies the power of 10 by which to divide the value. This argument is optional.
  - **Default:** 0
  - **Range:** 0–31

**Details**

Use S370FZDT\(w,d\) on other operating environments to read zoned decimal data from IBM mainframe files.

**Comparisons**

- Zoned decimal separate trailing-sign data are similar to zoned decimal separate leading-sign data except that the sign of the value is stored in the last byte of zoned decimal separate trailing-sign data.
- The S370FZDT\(w,d\) informat is equivalent to the COBOL notation PIC S9(\(n\)) DISPLAY SIGN TRAILING SEPARATE, where the \(n\) value is the number of digits.
Example

```plaintext
input @1 x s370fzd4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------</td>
<td>1</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.

---

### S370FZDU 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**

**w**

specifies the width of the input field.

- **Default:** 8
- **Range:** 1–32

**d**

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

- **Default:** 0
- **Range:** 0–31

---

**Details**

Use `S370FZDUw.d` 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.

SHRSTAMP Informat

Reads date and time values of SHR records.

Category: Date and Time

Syntax

```plaintext
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 `ccyydddFhmmsssth`, 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 \textit{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.

\textit{ddd} is the one-and-a-half bytes that contain three digits that correspond to the day of the year.

\textit{F} is the half-byte that contains all binary 1s.

\textit{hh} is the one-byte representation of two digits that correspond to the hour of the day.

\textit{mm} is the one-byte representation of two digits that correspond to minutes.

\textit{ss} is the one-byte representation of two digits that correspond to seconds.

\textit{th} is the one-byte representation of two digits that correspond to a 100th of a second.

The SHRSTAMP\textsubscript{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.

\textbf{Example}

\begin{verbatim}
input begin: $hex16.;
y=input(begin, shrstamp8.);
\end{verbatim}

\begin{tabular}{|c|c|}
\hline
\textbf{Data Line} & \textbf{Result} \\
\hline
0110239F12403576 & 1598532035.8 \\
\hline
\end{tabular}

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.

\textbf{SMFSTAMP Informat}

Reads time and date values of SMF records.

\textbf{Category:} Date and Time

\textbf{Syntax}

\texttt{SMFSTAMP\textsubscript{w}.}

\textbf{Syntax Description}

\texttt{w}

specifies the width of the input field.
Requirement  
W must be 8 because time and date values in SMF records contain eight bytes of information: four bytes of time data that are followed by four bytes of date data.

Tip  
The time portion of an SMF record is a four-byte integer binary number that represents time as the number of hundredths of a second past midnight.

Details  
The 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  
```plaintext
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.

**TIME 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

\texttt{TIME}_w. \\

**Syntax Description**

\texttt{w}

specifies the width of the input field.

**Default** \hspace{1cm} 8  \\

**Range** \hspace{1cm} 5–32  \\

**Details**

The \texttt{TIME}_w. informat reads SAS time values in the form: \texttt{hh:mm:ss<.ss> <AM | PM>}:  \\

\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 between 00 and 59 that represents minutes.  \\

\texttt{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.  \\

\texttt{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 \texttt{hh}, \texttt{mm}, and \texttt{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**

\begin{verbatim}
input begin time10.;
\end{verbatim}

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
<th>Formatted with \texttt{TIME}_w.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{12.56}</td>
<td>46560</td>
<td>12:56:00</td>
</tr>
<tr>
<td>\texttt{120:120}</td>
<td>439200</td>
<td>122:00:00</td>
</tr>
<tr>
<td>\texttt{1:13 pm}</td>
<td>47580</td>
<td>13:13:00</td>
</tr>
</tbody>
</table>
TODSTAMP Informat

Reads an eight-byte time-of-day stamp.

**Category:** Date and Time

**Syntax**

TODSTAMP<sup>w</sup>.

**Syntax Description**

<sup>w</sup> specifies the width of the input field.

**Requirement**

<sup>w</sup> must be 8 because the OS TIME macro or the STCK instruction on IBM mainframes each return an eight-byte value.

**Details**

The TODSTAMP<sup>w</sup> informat reads time-of-day clock values that are produced by IBM mainframe operating systems and converts the clock values to SAS datetime values. If the time-of-day value is all 0s, TODSTAMP<sup>w</sup> results in a missing value.

Use TODSTAMP<sup>w</sup> on other operating environments to read time-of-day values that are produced by an IBM mainframe.

**Example**

```sas
input btime: $hex16.;
y=input(btime, todstamp8.);
```
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.

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>B591183D5FB80000</td>
<td>1300786905</td>
</tr>
</tbody>
</table>

### TRAILSGN Informat

Reads a trailing plus (+) or minus (–) sign.

**Category:** Numeric

### Syntax

`TRAILSGNw.`

### Syntax Description

`w`

specifies the width of the input field.

_DEFAULT: 6_

_RANGE: 1–32_

### Details

If the data contains a decimal point, the TRAILSGN informat honors the number of decimal places that are in the input data. If the data contains a comma, the TRAILSGN informat reads the value, ignoring the comma.

### Example

```
input x trailsgn8.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1,000</td>
<td>1000</td>
</tr>
<tr>
<td>1+</td>
<td>1</td>
</tr>
<tr>
<td>1-</td>
<td>-1</td>
</tr>
</tbody>
</table>
TU Informat

Reads timer units.

Category: Date and Time

Syntax

TUw;

Syntax Description

w

specifies the width of the input field.

Requirement w must be 4 because the OS TIME macro returns a 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

input bttime tu4.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2</td>
<td>1.2</td>
</tr>
<tr>
<td>1.2+</td>
<td>1.2</td>
</tr>
<tr>
<td>1.2-</td>
<td>-1.2</td>
</tr>
</tbody>
</table>

The data line is a hexadecimal representation of a binary, four-byte timer unit value. Each byte occupies one column of the input field. The result is a SAS time value that corresponds to 5:26:58.41 p.m.
VAXRB 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:
- “RB Informat” on page 332

VMSZN 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

VMSZNw.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 VMSZNw.d informat is similar to the ZDw.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 VMSZNw.d informat and the ZDw.d informat is in the special character used for the last digit. The following table shows the special characters used by the VMSZNw.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.;
Data line | Result
---|---
| 1
1234 | 1234
123t | -1234

**See Also**

**Formats:**
- “VMSZN Format” on page 220

**Informats:**
- “ZD Informat” on page 367

---

**w 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>23</td>
<td>2300</td>
</tr>
<tr>
<td>23</td>
<td>2300</td>
</tr>
<tr>
<td>23</td>
<td>2300</td>
</tr>
<tr>
<td>23</td>
<td>-2300</td>
</tr>
<tr>
<td>23.0</td>
<td>23.</td>
</tr>
<tr>
<td>2.3E1</td>
<td>2.3</td>
</tr>
<tr>
<td>-23</td>
<td>0</td>
</tr>
</tbody>
</table>

YYMMDD Informat

Reads date values in the form \texttt{ymmmdd} or \texttt{yyyyymmmdd}.

**Category:** Date and Time

**Syntax**

\texttt{YYMMDDw}.

**Syntax Description**

\texttt{w}

specifies the width of the input field.

Default: 6

Range: 6–32
Details

SAS read date values in one of the following forms:

- **yy**
- **yyyy**

**** 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

```sas
input calendar_date yymmdd10.;
```

<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:**

- “DATE Format” on page 141
- “DDMMYY Format” on page 58
- “MMDDYY Format” on page 82
- “YYMMDD Format” on page 117

**Functions:**

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

Reads date values in the form yyyy_mm 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:

- yyyy_mm
- 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 yyymm6.;
### See Also

**Formats:**
- “DATE Format” on page 141
- “DDMMYY Format” on page 58
- “YYMMDD Format” on page 117
- “YYMM Format” on page 116
- “YYMON Format” on page 121

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

**Informats:**
- “DATE Informat” on page 293
- “DDMMYY Informat” on page 296
- “MMDDYY Informat” on page 318
- “YYMMDD Informat” on page 362

**System Options:**
- “YEARCUTOFF= System Option” in *SAS Viya System Options: Reference*

---

**YYQ Informat**

Reads quarters of the year in the form yyQq or yyyyQq.

**Category:** Date and Time

**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:

- \( yyQ \)
- \( yyyyQ \)

\( yy \) 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 \texttt{YEARCUTOFF=} system option.

Example

```plaintext
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 \textit{SAS Viya Functions and CALL Routines: Reference}
- “YEAR Function” in \textit{SAS Viya Functions and CALL Routines: Reference}
- “YYQ Function” in \textit{SAS Viya Functions and CALL Routines: Reference}

System Options:
- “\texttt{YEARCUTOFF=} System Option” in \textit{SAS Viya System Options: Reference}
**ZD 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.

---

**Syntax**

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

**Syntax Description**

\(w\)

specifies the width of the input field.

- Default: 1
- Range: 1–32

\(d\)

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

- Range: 1–31

---

**Details**

The `ZD_w.d` informat reads zoned decimal data in which every digit requires one byte and in which the last byte contains the value's sign along with the last digit.

*Note:* Different operating environments store zoned decimal values in different ways. However, `ZD_w.d` reads zoned decimal values with consistent results if the values are created in the same type of operating environment that you use to run SAS.

You can enter positive values in zoned decimal format from a personal computer. Some keying devices enable you to enter negative values by overstriking the last digit with a minus sign.

**Comparisons**

- Like the \(w.d\) informat, the `ZD_w.d` informat reads data in which every digit requires one byte. Use `ZDV_w.d` or `ZDw.d` to read zoned decimal data in which the last byte contains the last digit and the sign.

- The `ZD_w.d` informat functions like the `ZDVw.d` informat with one exception: `ZDVw.d` validates the input string and disallows invalid data.

- The following table compares the zoned decimal informat with notation in several programming languages:
### Zoned Decimal Notation

<table>
<thead>
<tr>
<th>Language</th>
<th>Zoned Decimal Notation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAS</td>
<td>ZD3.</td>
</tr>
<tr>
<td>PL/I</td>
<td>PICTURE'99T'</td>
</tr>
<tr>
<td>COBOL</td>
<td>DISPLAY PIC S 999</td>
</tr>
<tr>
<td>IBM assembler</td>
<td>ZL3</td>
</tr>
</tbody>
</table>

#### Example

```plaintext
input @1 x zd4.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>F0F1F2C8</td>
<td>128</td>
</tr>
</tbody>
</table>

* The data line contains a hexadecimal representation of a binary number that is stored in zoned decimal format on an IBM mainframe computer system. Each byte occupies one column of the input field.

#### See Also

**Informats**
- “w Informat” on page 361
- “ZDV Informat” on page 369

#### ZDB 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.

**Syntax**

`ZDB w.d`

**Syntax Description**

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

  Default: 1
Range 1–32

\( d \)

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

Range 0–31

Details

The ZDB\(w.d\) informat reads zoned decimal data that are produced in IBM 1410, 1401, and 1620 form, where 0s are left blank rather than being punched.

Example

```plaintext
input @1 x zdb3.;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</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.

ZDV 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

\[ ZDV^{w.d} \]

**Syntax Description**

- \( w \)
  - specifies the width of the input field.
  - Default 1
  - Range 1–32

- \( d \)
  - specifies the power of 10 by which to divide the value. This argument is optional.
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

Informat:

- “w Informat” on page 361
- “ZD Informat” on page 367
Informats by Category

<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>
</tbody>
</table>
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>Column Binary</td>
<td>ROW Informat (p. 392)</td>
<td>Reads a column-binary field down a card column.</td>
</tr>
<tr>
<td>Date and Time</td>
<td>B8601CI Informat (p. 374)</td>
<td>Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss&lt;fff&gt;.</td>
</tr>
<tr>
<td></td>
<td>B8601DA Informat (p. 375)</td>
<td>Reads date values that are specified using the ISO 8601 base notation yyyymmdd.</td>
</tr>
<tr>
<td></td>
<td>B8601DJ Informat (p. 376)</td>
<td>Reads a Java date and time value that is in the form yyyymmddhhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td></td>
<td>B8601DN Informat (p. 378)</td>
<td>Reads date values that are specified using the ISO 8601 basic notation yyyymmdd and returns SAS datetime values where the time portion of the value is 000000.</td>
</tr>
<tr>
<td></td>
<td>B8601DT Informat (p. 379)</td>
<td>Reads datetime values that are specified using the ISO 8601 basic notation yyyymmddThhhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td></td>
<td>B8601DZ Informat (p. 381)</td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation yyyymmddThhhmmss+–hhmm or yyyymmddThhhmmss&lt;ffffff&gt; Z.</td>
</tr>
<tr>
<td></td>
<td>B8601TM Informat (p. 383)</td>
<td>Reads time values that are specified using the ISO 8601 basic notation hhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td></td>
<td>B8601TZ Informat (p. 384)</td>
<td>Reads time values that are specified using the ISO 8601 basic time notation hhmmss&lt;ffffff&gt; +–hhmm or hhmmss&lt;ffffff&gt; Z.</td>
</tr>
<tr>
<td></td>
<td>E8601LZ Informat (p. 386)</td>
<td>Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation hh:mm:ss+–hh:mm or hh:mm:ss&lt;ffffff&gt; Z and converts the values to the local time.</td>
</tr>
<tr>
<td></td>
<td>PDJULG Informat (p. 389)</td>
<td>Reads packed Julian date values in the hexadecimal form yyyydddF for IBM.</td>
</tr>
<tr>
<td></td>
<td>PDJULI Informat (p. 390)</td>
<td>Reads packed Julian dates in the hexadecimal format ccyydddF for IBM.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>STIMER Informat (p. 393)</td>
<td>Reads time values and determines whether the values are hours, minutes, or seconds; reads the output of the STIMER system option.</td>
<td></td>
</tr>
<tr>
<td>WEEKU Informat (p. 394)</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>WEEKV Informat (p. 396)</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>WEEKW Informat (p. 398)</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>YMDDTTM Informat (p. 400)</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>ISO 8601</td>
<td>B8601CI Informat (p. 374)</td>
<td>Reads an IBM date and time value that includes a century marker, in the form cyymmdhhmmss&lt;fff&gt;.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>B8601DA Informat (p. 375)</td>
<td>Reads date values that are specified using the ISO 8601 base notation yyyyymmdd.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>B8601DJ Informat (p. 376)</td>
<td>Reads a Java date and time value that is in the form yyyyymmddhhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>B8601DN Informat (p. 378)</td>
<td>Reads date values that are specified using the ISO 8601 basic notation yyyyymmdd and returns SAS datetime values where the time portion of the value is 000000.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>B8601DT Informat (p. 379)</td>
<td>Reads datetime values that are specified using the ISO 8601 basic notation yyyyymmddThhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>B8601DZ Informat (p. 381)</td>
<td>Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation yyyyymmddThhmmss+</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>B8601TM Informat (p. 383)</td>
<td>Reads time values that are specified using the ISO 8601 basic notation hhmmss&lt;ffffff&gt;.</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>B8601TZ Informat (p. 384)</td>
<td>Reads time values that are specified using the ISO 8601 basic time notation hhmmss&lt;ffffff&gt;+</td>
</tr>
<tr>
<td>ISO 8601</td>
<td>E8601LZ Informat (p. 386)</td>
<td>Reads Coordinated Universal Time (UTC) values that are specified using the ISO 8601 extended notation hh:mm:ss+</td>
</tr>
<tr>
<td>Numeric</td>
<td>OCTAL Informat (p. 388)</td>
<td>Converts positive octal values to integers.</td>
</tr>
</tbody>
</table>
Dictionary

B8601CI Informat
Reads an IBM date and time value that includes a century marker, in the form cyymmddhhmmss<fff>.

Categories: Date and Time
ISO 8601

Alignment: Left

Syntax
B8601CI_{w.d}

Syntax Description

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

Default 16
Range 10–26

\( d \)
specifies the number of digits to the right of the decimal point in the seconds value.

Default 0
Range 0–6

Details

The B8601CI informat reads time values that are specified in the IBM time notation cyymmddhhmmss<fff>:

\( c \)
is a single digit that represents a century:

0 indicates the years 1900–1999.
1 indicates the years 2000–2099.
2 indicates the years 2100–2199.

\( n \)
indicates the years 00–99 in a century that is determined by performing a calculation on a year greater than 2199. To determine the century marker, subtract 1900 from the year and divide the result by 100. Discard the remainder. The remaining integer is the century marker. For example, to determine the century marker for the year 2382, perform this calculation: (2382–1900)/100=4.82. Discard .82. The century marker is 4.

\( yy \)
is a two-digit year between 00 and 99.
**mm**

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

**dd**

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

**hh**

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

**mm**

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

**ss**

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

**fff**

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

**Example**

```plaintext
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>

**B8601DA Informat**

Reads date values that are specified using the ISO 8601 base notation *yyyymmdd*.

**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 243

B8601DJ Informat

Reads a Java date and time value that is in the form yyyyMMddhhmmss<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 \( yyyyymmddhhmmss<ffffff> \):

- \( yyyy \) is a four-digit year between 0000 and 9999.
- \( mm \) is a two-digit month (zero padded) between 01 and 12.
- \( dd \) is a two-digit day of the month (zero padded) between 01 and 31.
- \( hh \) is a two-digit hour (zero padded) between 00 and 23.
- \( mm \) is a two-digit minute (zero padded) between 00 and 59.
- \( ss \) is a two-digit second (zero padded) between 00 and 59.
- \( fffffff \) are optional fractional seconds, with a precision of up to six digits, where each digit is between 0 and 9.

Comparisons

The B8601DJ informat reads a date and time value that does not include a T to separate the date from the time.

Java date and time values do not include a T. For example, the date January 1, 2011 at 4:30:25 a.m. is written as 20110101043025.

ISO 8601 date and time values include a T. For example, the date January 1, 2011 at 4:30:25 a.m. is written as 20110101T043025.

Example

input @1 bdj b8601dj.;
B8601DN Informat

Reads date values that are specified using the ISO 8601 basic notation \texttt{yyyymmdd} and returns SAS datetime values where the time portion of the value is 000000.

**Categories:** Date and Time

**ISO 8601**

**Alignment:** Left

**Alias:** ND8601DNw

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

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

### Syntax

\[ \texttt{B8601DN} \texttt{w}. \]

### Syntax Description

\texttt{w}

specifies the width of the input field.

**Default**

10

**Requirement**

The width of the input field must be 10.

### Details

The B8601DN informat reads date values that are specified using the ISO 8601 basic date notation \texttt{yyyymmdd} and returns the date in a SAS datetime value:

\texttt{yyyy}

is a four-digit year.

\texttt{mm}

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

\texttt{dd}

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

input @1 bdn b8601dn.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>----</td>
<td>1</td>
</tr>
<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 243

B8601DT Informat

Reads datetime values that are specified using the ISO 8601 basic notation `yyyyMMddThhmmss<ffffff>`.

**Categories:** Date and Time

ISO 8601

**Alignment:** Left

**Aliases:**

B8601LXw.d
ND8601DTw.d

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

**Syntax**

`B8601DT w.d`

**Syntax Description**

`w`

specifies the width of the input field.

Default 19

Range 19–26

`d`

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

Default 0

Range 0–6

**Details**

The B8601DT informat reads datetime values that are specified in the ISO 8601 basic datetime notation `yyyyMMddThhmmss<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>166343580</td>
<td>20120915T155300</td>
</tr>
<tr>
<td>2012</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 243

B8601DX 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 "B8601DZ Informat" on page 381.

Syntax

B8601DXw.
B8601DZ Informat

Reads Coordinated Universal Time (UTC) datetime values that are specified using the ISO 8601 datetime basic notation `yyyymmddThhmmss [+–] hhmm` or `yyyymmddThhmmss<ffffff>Z`.

**Categories:** Date and Time

**ISO 8601**

**Alignment:** Left

**Aliases:** B8601DXw.d

ND8601DZw.d

**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:

- `yyyymmddThhmmss [+–] hhmm`
- `yyyymmddThhmmss<ffffff>Z`

`yyyy`

is a four-digit year.

`mm`

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

`dd`

is a two-digit day of the month (zero padded) between 01 and 31.
hh is a two-digit hour (zero padded) between 00 and 24.

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

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

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 243

B8601LX 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 yyyyymmddThhmmss+|–hhmm.

See: The B8601LXw.d informat uses the B8601DTw.d informat to read data. See “B8601DT Informat” on page 379.

Syntax

B8601LXw.d
B8601TM Informat

Reads time values that are specified using the ISO 8601 basic notation *hhmms<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 *hhmms<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>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 243

B8601TX 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 \texttt{hhmmss+|–hhmm}.

See: The \texttt{B8601TXw} informat uses the \texttt{B8601TZw.d} informat to read data. See “\texttt{B8601TZ Informat}” on page 384.

Syntax

\texttt{B8601TXw.}

B8601TZ Informat

Reads time values that are specified using the ISO 8601 basic time notation \texttt{hhmmss<ffff>+|–hhmm} or \texttt{hhmmss<ffffff>Z}.

Categories: Date and Time
ISO 8601

Alignment: Left

Aliases: \texttt{B8601TXw.d}
\texttt{ND8601TZw.d}

Restriction: UTC time zone offset values are not supported.

Supports: ISO 8601 Element 5.3.1.1, complete representation

Syntax

\texttt{B8601TZw.d}

Syntax Description

\texttt{w}

specifies the width of the input field.
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:

- \( hh:mm:ss<ffffff>+|–hh:mm \)
- \( hh:mm:ss<ffffff>Z \)

\( hh \)
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.

\( fffffff \)
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 four characters).

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

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

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

When SAS reads a UTC time by using the B8601TZ informat and the adjusted time is greater than 240000 or less than 000000, SAS adjusts the time value so that the time is between 000000 and 240000. For example, if SAS reads the UTC time 234344–0500 using the B8601TZ informat, SAS adds five hours to the time so that the value is 284344, and then makes the time adjustment. The value stored represents the time 044344+0000.

Example

input @1 btz b8601tz.;
<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>202401-0500</td>
<td>5041</td>
</tr>
<tr>
<td>202401Z</td>
<td>73441</td>
</tr>
<tr>
<td>202401+0000</td>
<td>73441</td>
</tr>
</tbody>
</table>

**See Also**

“Reading Dates and Times By Using the ISO 860 Basic and Extended Notations” on page 243

---

**E8601LX 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 “E8601DT Informat” on page 300.

**Syntax**

```
E8601LXw.
```

---

**E8601LZ 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
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

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:

- \( hh:mm:ss.ffffff \pm hh:mm \)
- \( hh:mm:ss.ffffff Z \)

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

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

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

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

\( \pm hh:mm \) 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 E8601LZ informat and the adjusted time is greater than 24:00:00 or less than 00:00:00, SAS adjusts the value so that the time is between 00:00:00 and 24:00:00. For example, if SAS reads the UTC time 23:43:44-05:00 by using the E8601LZ informat, SAS adds five hours to the time so that the value is 28:43:44, and then makes the time adjustment. The value stored represents the time 04:43:44+00:00.

**Example**

```
  input elz e8601lz.;
```
See Also

“Reading Dates and Times By Using the ISO 8601 Basic and Extended Notations” on page 243

---

E8601TX 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 \textit{hh:mm:ss}|-\textit{hh:mm}.

See: The E8601TX\textit{w}. informat uses the E8601TZ\textit{d} informat to read data. See “E8601DZ Informat” on page 302.

Syntax

\texttt{E8601TX\textit{w}.}

---

OCTAL Informat

Converts positive octal values to integers.

Category: Numeric

Syntax

\texttt{OCTAL\textit{w}.\textit{d}}

\textbf{Syntax Description}

\textit{w}

specifies the width of the input field.

Default 3

Range \texttt{1–24}

\textit{d}

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

Range \texttt{1–31}

Restriction must be greater than or equal to the \textit{w} value.
Details

Use only the digits 0 through 7 in the input, with no embedded blanks. The OCTALw.d informat ignores leading and trailing blanks.

OCTALw.d cannot read negative values. It treats all input values as positive (unsigned).

Example

```plaintext
input @1 value octal3.1;
```

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>---------1</td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>12.7</td>
</tr>
</tbody>
</table>

PDJULG 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:
- “JULDAY Format” on page 188
- “JULIAN Format” on page 80
- “PDJULG Format” on page 191
- “PDJULI Format” on page 193

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

Informats:
- “JULIAN Informat” on page 315
- “PDJULI Informat” on page 390

System Options:
- “YEARCUTOFF= System Option” in SAS Viya System Options: Reference

PDJULI Informat

Reads packed Julian dates in the hexadecimal format ccyydddF for IBM.

**Category:** Date and Time

**Syntax**

```
PDJULIw.
```

**Syntax Description**

w specifies the width of the input field.

Default 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.;

<table>
<thead>
<tr>
<th>Data Line</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>0099001F</td>
<td>14245</td>
</tr>
<tr>
<td>0112015F</td>
<td>19007</td>
</tr>
</tbody>
</table>

* SAS date value 14245 is January 1, 1999. SAS date value 19007 is January 15, 2012.

See Also

Formats:
• “JULDAY Format” on page 188
• “JULIAN Format” on page 80
• “PDJULG Format” on page 191
• “PDJULI Format” on page 193

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

Informats:
• “JULIAN Informat” on page 315
ROW Informat

Reads a column-binary field down a card column.

**Category:** Column Binary

**Syntax**

ROW\textsubscript{w.d}

**Syntax Description**

\textit{w} specifies the row where the field begins.

- **Range:** 0–12

\textit{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\textsubscript{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\textsubscript{w.d} assigns the variable a missing value and sets the automatic variable \_ERROR\_ to 1. If the field has no punches, then ROW\textsubscript{w.d} assigns the variable a missing value.

ROW\textsubscript{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\textsubscript{w.d} reads a field, the pointer moves to the next row.

**Example**

\texttt{input x row5.3}
\texttt{input x row7.1}
\texttt{input x row5.2}
\texttt{input x row3.5}
The data line is a hexadecimal representation of the column binary. The “virtual” punch card column for the example data has row 7 punched. The binary representation is 0000 0000 0000 0100.

**See Also**

**Informats:**
- “$CB Informat” on page 263
- “CB Informat” on page 290
- “PUNCH Informat” on page 331

**STIMER Informat**

Reads time values and determines whether the values are hours, minutes, or seconds; reads the output of the STIMER system option.

**Category:** Date and Time

**Syntax**

STIMER<sub>w</sub>.  

**Syntax Description**

<sub>W</sub>  

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.

---

**WEEKU Informat**

Reads a value in the form of a week-number within the year and returns a SAS date value by using the U algorithm.

**Category:** Date and Time

**Syntax**

`WEEKUw.`

**Syntax Description**

`w`

specifies the width of the input field.

Default 11

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>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;</td>
<td>18993 18993 18993 18993 18993</td>
</tr>
</tbody>
</table>

**See Also**

**Formats:**

- “WEEKU Format” on page 110
WEEKV Informat

Reads a value in the form a week-number within a year and returns a SAS date value using the V algorithm.

**Category:** Date and Time

**Syntax**

WEEKV <w>.

**Syntax Description**

`<w>`

specifies the width of the input field.

**Default**

11

**Range**

3–200

**Details**

The 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>
</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>---+----1</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
  v &= \text{input}('W01',\text{weekv1.}); \\
  w &= \text{input}('03W01',\text{weekv5.}); \\
  x &= \text{input}('03W0101',\text{weekv7.}); \\
  y &= \text{input}('2003W0101',\text{weekv9.}); \\
  z &= \text{input}('2003-W01-01',\text{weekv11.}); \\
  \text{put } v; \\
  \text{put } w; \\
  \text{put } x; \\
  \text{put } y; \\
  \text{put } z;
\end{align*}
\]

\[
\begin{align*}
  &18994 \\
  &18994 \\
  &18994 \\
  &18994 \\
  &18994 \\
\end{align*}
\]

See Also

Formats:
- “WEEKU Format” on page 110
- “WEEKV Format” on page 111
- “WEEKW Format” on page 113

Functions:
**WEEKW 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**

\[ \text{WEEKW} \_w. \]

**Syntax Description**

\( w \)

specifies the width of the input field.

**Default** 11

**Range** 3–200

**Details**

The WEEKW. informat reads the week-number value within the year. If the input does not contain a year expression, the WEEKW. informat uses the current year as the year expression, which is the default. If the input does not contain a day expression, the WEEKW. informat uses the first day of the week as the day expression, which is the default. Algorithm W calculates the SAS date value using the number of the week within the year (Monday is considered the first day of the week). The number-of-week value is represented as a decimal number in the range 0–53, with a leading zero and maximum value of 53. For example, the fifth week of the year would be represented as 05.

The inputs to the WEEKW. 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 WEEKW informat reads the week-number value as a decimal number in the range 00–53, with Monday as the first day of week.

The WEEKU informat reads the week-number value as a decimal number in the range 00–53, with Sunday as the first day of the week.

The WEEKV 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>v=input('W01',weekw3.);</td>
<td>18994</td>
</tr>
<tr>
<td>w=input('03W01',weekw5.);</td>
<td>18994</td>
</tr>
<tr>
<td>x=input('03W0101',weekw7.);</td>
<td>18994</td>
</tr>
<tr>
<td>y=input('2003W0101',weekw9.);</td>
<td>18994</td>
</tr>
<tr>
<td>z=input('2003-W01-01',weekw11.);</td>
<td>18994</td>
</tr>
<tr>
<td>put v;</td>
<td></td>
</tr>
<tr>
<td>put w;</td>
<td>18994</td>
</tr>
<tr>
<td>put x;</td>
<td>18994</td>
</tr>
<tr>
<td>put y;</td>
<td>18994</td>
</tr>
<tr>
<td>put z;</td>
<td>18994</td>
</tr>
</tbody>
</table>

See Also

Formats:
- “WEEKU Format” on page 110
- “WEEKV Format” on page 111
- “WEEKW Format” on page 113

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

Informats:
- “WEEKU Informat” on page 394
- “WEEKV Informat” on page 396
YMDDTTM 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.

<table>
<thead>
<tr>
<th>Category:</th>
<th>Date and Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alignment:</td>
<td>Right</td>
</tr>
</tbody>
</table>

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>:<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.
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 in format reads datetime values with optional separators in the form mm-dd-yy<yy> hh:mm:ss.ss AM | PM, and requires a space between the date and the time.

The 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

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:
- “DATETIME Informat” on page 295
- “MDYAMPM Informat” on page 316
Recommended Reading

Here is the recommended reading list for this title:

- *An Array of Challenges--Test Your SAS Skills*
- *Base SAS Glossary*
- *SAS Viya Utility Procedures Guide*
- *Batch and Line Mode Processing in SAS Viya*
- *The Little SAS Book, A Primer, Fifth Edition*
- *The SAS Workbook*

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
Web address: [sas.com/store/books](http://sas.com/store/books)
Index

Special Characters
SASCIIw. format 139
SASCIIw. informat 260
$BASE64Xw. format 37
$BASE64Xw. informat 261
$BINARyw. format 38
$BINARyw. informat 262
$CBw. informat 263
SCHARw. format 39
SCHARw. informat 264
compared to $ASCII informat 260
compared to $CHARZBw. informat 266
compared to $SEBCDICw. informat 267
compared to $w. informat 278
SCHARZBw. informat 265
SEBCDICw. format 143
SEBCDICw. informat 266
compared to S370FFw.d informat 337
$Fw. format 46
$Fw. informat 278
$HEXw. format 40
$HEXw. informat 267
compared to $BINARyw. informat 262
$MSGCASEw. format 144
$N8601BAw.d format 146
$N8601Bw.d format 145
$N8601Bw.d informat 268
$N8601EAw.d format 149
$N8601EHw.d format 150
$N8601Ew.d format 147
$N8601Ew.d informat 270
$N8601EXw.d format 151
$N8601Hw.d format 153
$N8601Xw.d format 154
SOCTALw. format 41
SOCTALw. informat 272
SPHEXw. informat 273
$QUOTEw. format 42
$QUOTEw. format 274
$REVERJw. format 43
$REVERSew. format 44
$SUPCASEw. format 45
$SUPCASEw. informat 275
$VARYINGw. format 155
$VARYINGw. informat 276
$w. format 46
$w. informat 278
%SYSFUNC function
specifying formats with 5

A
AM or PM
datetime values with 54, 316
time values with 100
ANYDTEw. informat 279
ANYDTDMw. informat 281
ANYDTTMEw. informat 285
ASCII
converting character data to 139
ASCII data
converting character data to, Base 64
encoding 37, 261
converting to native format 260
ATTRIB statement
specifying formats with 6
specifying informats with 236

B
B8601CI informat 374
B8601Aw. format 157
B8601Aw. informat 375
B8601DJ informat 376
B8601DNw. format 158
B8601DNw. informat 378
B8601Dw.d format 159
B8601Dw.d informat 379
B8601Xw. format 161
B8601Xw. informat
See B8601DZw.d informat
B8601DXw.d informat 381
B8601DXw. format 162
B8601DXw.d informat 381
B8601LXw. format 164
B8601LXw. informat
See B8601DTw.d informat
B8601LXw.d informat 379
B8601LZw. format 165
B8601TMw.d format 166
B8601TMw.d informat 383
B8601TXw. format 168
B8601TXw. informat
See B8601TZw.d informat
B8601TXw.d informat 384
B8601TZw.d format 169
B8601TZw.d informat 384
Base 64 encoding
converting character data to ASCII text 37, 261
BASE64X 261
BESTDOTXw. format 50
BESTDw.p format 48
BESTw. format 46
BESTw.d informat 361
big endian platforms
byte ordering 8
big endian platforms, byte ordering on
binary
converting character data to 38
converting numeric values to 51
binary data, converting to
character 262
integers 287
binary zeros, converting to blanks 265
BINARYw. format 51
BINARYw.d informat 287
bits, extracting 288
BITSw.d informat 288
blanks
converting binary zeros to 265
converting to zeros 289
byte ordering 8, 237
BZw.d informat 289
compared to w.d informat 362
writing 39, 46
writing in uppercase 144
character data, reading
from column-binary files 263
standard format 278
varying length fields 276
with blanks 263
column-binary data, reading
down a column 392
punch-card code 331
column-binary files, reading 263
column-binary, reading
with blanks 264
commas
in numeric values 52, 53
replacing decimal points with 189
COMMAw.d format 52
COMMAw.d informat 291
compared to COMMAXw.d informat 293
COMMAXw.d format 53
COMMAXw.d informat 292
compared to COMMAw.d informat 292
D
data conversion
formats and 9
data values, reading 233
date and time informats
B8601DN informat, ISO 8601 basic
date notation, returns the date in a
datetime value 378
B8601DT informat, ISO 8601 basic
datetime notation, no time zone 379
reading IBM dates and times 374
reading Java dates and times 376
date and time values
SHR records 352
date values
as day of month 58
B8601DA format, ISO 8601 basic
notation 157
B8601DA informat, ISO 8601 basic
notation 375
DATEw. format 141
day-of-week name 64
DDMMYYw. format 58
DDMMYYxw. format 60
DTDATEw. format 65
E8601DA format, ISO 8601 extended
notation 171
E8601DA informat, extended notation 298
C
character data
converting to ASCII 139
converting to ASCII text, Base 64
encoding 37, 261
converting to binary 38
converting to EBCDIC 143
converting to hexadecimal 43
converting to octal 41
reverse order, left alignment 44
reverse order, preserving blanks 43
uppercase conversion 45
varying length 155
E8601DN informat, ISO 8601 extended notation, returns date in datetime value 299
extracting from informat values 279
Julian dates 80
Julian day of the year 188
MMDDYYw. format 82
MMDDYYxw. format 84
MMYYw. format 88
MMYYxw. format 89
month name 91
month of the year 92
MONYYw. format 92
quarter of the year 96
quarter of the year in Roman numerals 97
WEEKDATEw. format 106
WEEKDATXw. format 107
WEEKDAYw. format 109
WORDDATEw. format 221
WORDDATXw. format 223
YEARw. format 115
YYMMDDw. format 117
YYMMDDxw. format 119
YYMMw. format 116
YYMMxw. format 226
YYMONw. format 121
YYQRw. format 125
YYQRxw. format 126
YYQw. format 122
YYQxw. format 123
date/time values, reading
date values, ddmmyy 293
date values, ddmmyyy hh:mm:ss 295
date values, ddmmyyy yyyy 293
date values, ddmmyyyyy hh:mm:ss 295
date values, dmmmyy 296
date values, dmmmyyyy 296
date, yymm 364
date, yymmn 364
dates, mmdy 318
dates, mmdy yyyy 318
dates, yymmd 362
dates, yyyymmd 362
IBM mainframes 325
IBM mainframes, RMF records 334
IBM mainframes, SMF records 353
Julian dates 315
month and year values 319
RMF records 325
SMF records 325
TIME MIC values 321
time values, IBM mainframe 321
time-of-day stamp 356
time, hh:mm:ss ss 354
timer units 358
year quarter 365
DATEAMPMw.d format 54
datetime values
$N8601EA format, ISO 8601 extended notation 149
datetime formats
ISO 8601 extended datetime with no time zone 173
ISO 8601 extended datetime, with time zone 176
datetime informats
B8601DZ informat, ISO 8601 basic notation with time zone 381
datetime values
$N8601 informat, ISO 8601 basic and extended notation 268
$N8601B format, basic notation 145
$N8601BA format, ISO 8601 basic notation 146
$N8601E format, extended notation 147
$N8601E informat, extended notation 270
$N8601EH format, ISO 8601 extended notation, hyphen for omitted components 150
$N8601EX format, extended notation, x for omitted components 151
$N8601H format, basic notation, hyphen for omitted components 153
$N8601X format, x for omitted components 154
B8601DN format, ISO 8601 basic datetime notation, formats the date 158
B8601DT format, ISO 8601 basic notation, no time zone 159, 161, 164
B8601DZ format, ISO 8601 basic notation with time zone 162
B8601TX format, ISO 8601 basic notation, time zone offset 168
DATEAMPMw.d format 54
DATETIMEw.d format 56
DTDATEx. format 65
DTMONYYw. format 66
DTWKDATxw. format 68
DTYEARw. format 69
DYYQCw. format 70
E8601DN format, ISO 8601 extended, formats the date 172
E8601DT format, ISO 8601 basic notation, no time zone 174, 177
E8601DT informat, ISO 8601 extended notation, no time zone 300
E8601DZ informat, ISO 8601 extended notation with time zone 302
E8601LZ informat, ISO 8601 extended local notation with time zone 386
E8601TX format, ISO 8601 basic notation, time zone offset 182
extracting from informat values 281
with AM or PM 54
datetime values
  YMDDTTMw.d format 400
  DATETIMEw. informat 295
  DATETIMEw.d format 56
  DATEw. format 141
  DATEw. informat 293
  DAYw. format 58
  DDMYYYw. format 58
  DDMYYYw. informat 296
  DDMYYYxw. format 60
  DEC format
    integer binary (fixed-point) values in 186
    positive integer binary (fixed-point) values in 198
    reading integer binary values in 312
    reading positive integer binary values in 328
decimal places
  aligned 48, 140
decimal points
  replacing with commas 189
decimal points, reading as commas 322
  DOLLARw.d format 62
  DOLLARw.d informat 291
  DOLLARXw.d format 63
  DOLLARXw.d informat 292
double quotation marks
  data values in 42
  DOWNNAMEw. format 64
  DTDATEw. format 65
  DTMONYYw. format 66
  DTWKDATXw. format 68
  DTYEARw. format 69
  DTYYQCw. format 70
duration values
  $N8601 informat, ISO 8601 basic and extended notation 268
duration values
  $N8601B format, basic notation 145
  $N8601BA format, ISO 8601 basic notation 146
  $N8601E format, extended notation 147
  $N8601E informat, extended notation 270
$N8601EA format, ISO 8601 extended notation 149
$N8601EH format, ISO 8601 extended notation, hyphen for omitted components 150
$N8601EX formats, extended notation, x for omitted components 151
$N8601H format, basic notation, hyphen for omitted components 153
$N8601X format, x for omitted components 154
Dw.d format 361
Dw.p format 140

E
E8601DAw. format 171
E8601Daw. format 298
E8601DNw. format 172
E8601DNw. informat 299
E8601DTw.d format 173
E8601DTw.d format 300
E8601DXw. format 174
E8601DXw. informat
  See E8601DZw.d informat
E8601DXw.d format 302
E8601DZw. format 176
E8601DZw.d format 302
E8601LXw. format 177
E8601LXw. informat
  See E8601DTw.d informat
E8601LXw.d format 300
E8601LZw. format 179
E8601LZw.d format 306
E8601TMw.d format 180
E8601TMw.d format 304
E8601TXw. format 182
E8601TXw. informat
  See E8601TZw.d informat
E8601TXw.d format 305
E8601TZw. format 183
E8601TZw.d format 305
EBCDIC
  converting character data to numeric data in 204
EBCDIC data
  convert to native format 266
reading 336
embedded characters, removing 291, 292
encoding
  formats and 9
Ew. format 71
Ew.d format 361
fixed-point values
   DEC format  186, 198
   Intel format  186, 198
   reading in Intel and DEC formats  312, 328
   writing  185, 197
floating-point data (IEEE), reading  313
floating-point data, reading  307
floating-point values  72
   IEEE  78
FLOATw.d format  72
FLOATw.d informat  307
FORM 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, 243
   fractions  73, 224
FRACTw. format  73
   Fw.d format  105
   Fw.d informat  361

IBM
   packed Julian dates in hexadecimal for  193
IBM dates and times, reading  374, 376
IBM mainframe format
   integer binary (fixed-point) values in  205
   numeric data in  204
   packed decimal data in  208
   positive integer binary (fixed-point) values in  210
   real binary (floating-point) data in  212
   unsigned integer binary (fixed-point) values in  206
   unsigned packed decimal data in  209
   unsigned zoned decimal data in  218
   zoned decimal data  213
   zoned decimal leading-sign data in  214
   zoned decimal separate leading-sign data in  215
   zoned decimal separate trailing-sign data in  216
IBM packed decimal data, reading  323
   IBRw.d format  186
   IBRw.d informat  312
   IBw.d format  185
   IBw.d informat  311
   compared to S370FIBw.d informat  338
IEEE floating-point values  78
   reading  313
   IEEEw.d format  78
   IEEEw.d informat  313
   INFORMAT statement
      specifying informats with  235
   informats  233
      byte ordering  237
      integer binary notation  238
      packed decimal data and  12
      permanent  236
      specifying, with ATTRIB statement  236
      specifying, with INFORMAT statement  235
      specifying, with INPUT function  235
hexadecimal
   converting character data to  40
   converting real binary (floating-point) values to  74
   packed Julian dates in  191
   packed Julian dates in, for IBM  193
   hexadecimal binary values, converting to integers  308
   hexadecimal binary values, converting to real binary  308
   hexadecimal data, converting to character  267
   hexadecimal values
reading packed Julian date values in, for IBM  389
reading packed Julian dates in, for IBM  390
   HEXw. format  74
   HEXw. informat  308
   compared to $HEXw. informat  268
   IHHMSSw. informat  309
   IHHMMw.d format  75
   HOURw.d format  77

\section*{Index}

\textbf{F}
fixed-point values
   DEC format  186, 198
   Intel format  186, 198
   reading in Intel and DEC formats  312, 328
   writing  185, 197
floating-point data (IEEE), reading  313
floating-point data, reading  307
floating-point values  72
   IEEE  78
FLOATw.d format  72
FLOATw.d informat  307
\textbf{FORM} 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, 243
   fractions  73, 224
FRACTw. format  73
   Fw.d format  105
   Fw.d informat  361

\section*{\textbf{H}}
\textbf{HEX}  40
hexadecimal
   converting character data to  40
   converting real binary (floating-point) values to  74
   packed Julian dates in  191
   packed Julian dates in, for IBM  193
   hexadecimal binary values, converting to integers  308
   hexadecimal binary values, converting to real binary  308
   hexadecimal data, converting to character  267
   hexadecimal values
reading packed Julian date values in, for IBM  389
reading packed Julian dates in, for IBM  390
   HEXw. format  74
   HEXw. informat  308
   compared to $HEXw. informat  268
   IHHMSSw. informat  309
   IHHMMw.d format  75
   HOURw.d format  77

\textbf{I}
IBM
   packed Julian dates in hexadecimal for  193
IBM dates and times, reading  374, 376
IBM mainframe format
   integer binary (fixed-point) values in  205
   numeric data in  204
   packed decimal data in  208
   positive integer binary (fixed-point) values in  210
   real binary (floating-point) data in  212
   unsigned integer binary (fixed-point) values in  206
   unsigned packed decimal data in  209
   unsigned zoned decimal data in  218
   zoned decimal data  213
   zoned decimal leading-sign data in  214
   zoned decimal separate leading-sign data in  215
   zoned decimal separate trailing-sign data in  216
IBM packed decimal data, reading  323
   IBRw.d format  186
   IBRw.d informat  312
   IBw.d format  185
   IBw.d informat  311
   compared to S370FIBw.d informat  338
IEEE floating-point values  78
   reading  313
   IEEEw.d format  78
   IEEEw.d informat  313
   INFORMAT statement
      specifying informats with  235
   informats  233
      byte ordering  237
      integer binary notation  238
      packed decimal data and  12
      permanent  236
      specifying, with ATTRIB statement  236
      specifying, with INFORMAT statement  235
      specifying, with INPUT function  235
specifying, with INPUT statement 235
syntax 234
temporary 236
user-defined 236
zoned decimal data and 12
INPUT function
specifying informats with 235
INPUT statement
specifying informats with 235
integer binary (fixed-point) values
IBM mainframe format 205
integer binary data
byte ordering 8
notation and programming languages 9
integer binary data, reading
IBM mainframe format 337, 342
integer binary notation 238
integer binary values
DEC format 186
Intel format 186
reading in Intel and DEC formats 312
writing 185
integer binary values, reading 311, 326
integers
printing without decimals 48
Intel format
integer binary (fixed-point) values in 186
positive integer binary (fixed-point) values in 198
reading integer binary values in 312
reading positive integer binary values in 328
interval values
$N8601BA format, ISO 8601 basic notation 146
$N8601EA format, ISO 8601 extended notation 149
$N8601EX formats, extended notation, x for omitted components 151
$N8601H format, basic notation, hyphen for omitted components 153
interval values
$N8601 informat, ISO 8601 basic and extended notation 268
$N8601B format, basic notation 145
$N8601E format, extended notation 147
$N8601E informat, extended notation 270
$N8601EH format, ISO 8601 extended notation, hyphen for omitted components 150
$N8601X format, x for omitted components 154
ISO 8601 date and time formats
B8601DA format, basic date notation 157
B8601DN format, basic datetime notation, formats the date 158
B8601DT format, basic datetime notation, no time zone 159
B8601DX format, basic datetime notation, time zone offset 161
B8601DZ format, basic datetime notation with time zone 162
B8601EX format, basic datetime notation, time zone offset 164
B8601LX format, basic datetime notation, no time zone 165
B8601TM format, basic local time with time zone 166
B8601TX format, basic time notation, time zone offset 168
B8601TZ format, basic time notation with time zone 169
E8601DA format, extended date notation 171
E8601DN format, extended datetime notation, formats the date 172
E8601DX format, basic datetime notation, time zone offset 174
E8601LX format, basic datetime notation, time zone offset 177
E8601TM format, extended time notation, no time zone 180
E8601TX format, basic time notation, time zone offset 182
E8601TZ format, extended time notation with time zone 183
extended datetime, with time zone 176
extended local time with UTC offset 179
ISO 8601 date and time informats
$N8601 informat, basic and extended notation for durations, datetimes, and intervals 268
Index 411

$N8601E informat, extended notation for duration, datetime, and interval 270
B8601DA informat, basic date notation 375
B8601DN informat, basic datetime notation, returns the date in a datetime value 378
B8601DZ informat, basic datetime notation with time zone 381
B8601TM informat, basic time notation, no time zone 383
B8601TZ informat, basic time notation with time zone 384
E8601DA informat, extended date notation 298
E8601DN informat, extended notation, returns date in a datetime value 299
E8601DT informat, basic datetime notation, no time zone 379
E8601DT informat, extended datetime notation, no time zone 300
E8601DZ informat, extended datetime notation with time zone 302
E8601LZ informat, extended local datetime notation with time zone 386
E8601TM informat, extended time notation, no time zone 304
E8601TZ informat, extended time notation with time zone 305
ISO 8601 datetime formats
extended datetime with no time zone 173
ISO 8601 duration and datetime formats
$N8601B format, basic notation 145
$N8601BA format, basic notation 146
$N8601E format, extended notation 147
$N8601EA format, extended notation 149
$N8601EH format, extended notation, hyphen for omitted components 150
$N8601H format, hyphen for omitted components 153
$N8601X format, x for omitted components 154
ISO 8601 formatting symbols 14, 243
ISO 8602 duration and datetime formats
$N8601EX formats, extended notation, x for omitted components 151

Julian date values, packed reading in hexadecimal form, for IBM 389
Julian dates 80, 240
day of the year 188
packed 11
packed values in hexadecimal 191
packed values in hexadecimal for IBM 193
Julian dates, packed reading in hexadecimal format, for IBM 390
JULIANw. format 80
JULIANw. informat 315

L
leading zeros 128
little endian platforms
byte ordering 8
little endian platforms, byte ordering on 237

M
MDYAMPw.d format 81
MDYAMPw.d informat 316
MicroFocus COBOL
zoned numeric data 359
MicroFocus Cobol zoned numeric data 220
minus sign
trailing 357
MMDDYYw. format 82
MMDDYYw. informat 318
MMDDYYxw. format 84
MMSSw.d format 86
MMYYw. format 88
MMYYxw. format 89
MONNAMEw. format 91
MONTHw. format 92
MONYYw. format 92
MONYYw. informat 319
MSECw. informat 321

N
N8601B 145
N8601E 147
N8601EH 150
ND8601DAw. informat 375
ND8601DNw. informat 378
ND8601DTw.d informat 379
ND8601DZw.d informat 381
ND8601TMw.d informat 383
ND8601TZw.d informat 384

J
Java dates and times, reading 376
JULDAYw. format 188
negative numeric values
writing in parentheses 94
NEGPARENw.d format 94
nibble 239
definition 10
numeric data
commas replacing decimal points 189
EBCIDC format 204
IBM mainframe format 204
leading zeros with 128
one digit per byte 105
scientific notation 71
zoned decimal format 227
numeric data, reading
commas for decimal points 322
from column-binary files 290
standard format 361
numeric values
aligning decimal places 48, 140
best notation 46
commas in 52, 53
converting to binary 51
converting to fractions 73
converting to octal 95
DOLLARw.d format 62
DOLLARXw.d format 63
Roman numerals 97
words with numeric fractions 224
writing as percentages 194
writing as words 225
writing negative values in parentheses 94
NUMXw.d format 189
NUMXw.d informat 322
definition 10
formats and 10
formats and informats for 242
IBM mainframe format 208
languages supporting 12, 241
platforms supporting 11, 241
summary of formats and informats 12
unsigned format 200
packed decimal format
writing data in 190
packed hexadecimal data, converting to character 273
packed Julian date values
reading in hexadecimal form, for IBM 389
writing in hexadecimal 191
writing in hexadecimal for IBM 193
packed Julian dates 11, 240
reading in hexadecimal format, for IBM 390
parentheses
writing negative numeric values in 94
PDJULGw. format 191
PDJULGw. informat 389
PDJULiw. format 193
PDJULiw. informat 390
PDTIMEw. informat 325
compared to RMFSTAMPw. informat 335
PDw.d format 190
PDw.d informat 323
compared to $HXw. informat 274
compared to PKw.d informat 330
compared to S370FPDw.d informat 341
percentages
converting to numeric values 326
numeric values as 194
with minus sign for negative values 195
PERCENTNw.d format 195
PERCENTw.d format 194
PERCENTw.d informat 326
permanent formats 6
PIBrw.d format 198
PIBrw.d informat 328
PIBw.d format 197
PIBw.d informat 326
compared to S370FPIBw.d informat 343
PKw.d format 200
PKw.d informat 330
plus sign
trailing 357
PM or AM
datetime values with 54
time values with 100

O
octal
converting character data to 41
converting numeric values to 95
octal data
converting to character 272
converting to integers 388
OCTALw. format 95
OCTALw. informat
compared to SOCTALw. informat 273
OCTALw.d informat 388

P
p-values
writing 201
packed data, reading in IBM mainframe format 340
packed decimal data 10, 240
defined 239

Index
positive integer binary (fixed-point) values
  IBM mainframe format 210
positive integer binary values
  DEC format 198
  Intel format 198
  reading in Intel and DEC formats 328
  writing 197
programming languages
  integer binary notation and 9
  packed decimal data support 12
  zoned decimal data support 12
PUNCH.d informat 331
PUT function
  specifying formats with 5
PUT statement
  specifying formats with 5
PVALUEw.d format 201

Q
QTRRw. format 97
QTRw. format 96
  quotation marks
    removing 274

R
RBw.d format 202
RBw.d informat 332
  compared to S370FRBw.d informat 344
  compared to VAXRBw.d informat 359
reading data values 233
real binary (floating-point) data
  IBM mainframe format 212
  VMS format 104
real binary (floating-point) values
  converting to hexadecimal 74
real binary data
  real binary format 202
real binary data, reading 332
  IBM mainframe format 344
  VMS format 359
real binary format
  real binary data (floating-point) in 202
  reverse order character data 43, 44
RMF records, reading duration intervals 333
RMFDURw. informat 333
RMFSTAMPw. informat 334
  compared to RMFDURw. informat 334
roman numerals 125, 126
Roman numerals 97
ROMANw. format 97
ROWw.d informat 392

S
S370FFw.d format 204
S370FFw.d informat 336
S370FIBUw.d format 206
S370FIBUw.d informat 338
S370FIBw.d format 205
S370FIBw.d informat 337
S370FPDUw.d format 209
S370FPDUw.d informat 341
S370FPDw.d format 208
S370FPDw.d informat 340
  compared to S370FPDUw.d informat 342
S370FPIBw.d format 210
S370FPIBw.d informat 342
  compared to S370FIBUw.d informat 339
S370FRBw.d format 212
S370FRBw.d informat 344
S370FZDB 346
S370FZDBw.d informat 346
S370FZDLw.d format 214
S370FZDLw.d informat 347
S370FZDSw.d format 215
S370FZDSw.d informat 349
S370FZDTw.d format 216
S370FZDTw.d informat 350
S370FZDUw.d format 351
S370FZDUw.d informat 350
S370FZDw.d format 213
S370FZDw.d informat 345
  compared to S370FZDUw.d informat 352
SAS informats 233
scientific notation 71
SHR records
  reading data and time values of 352
SHRSTAMPw. informat 352
SMFSTAMPw. informat 353
Social Security numbers 219
SSNw. format 219
STIMERw. format 393

T
temporary formats 6
time values
  B8601LZ format, ISO 8601 basic local time with time zone 165
  B8601TM format, ISO 8601 basic time notation, no time zone 166
  B8601TM informat, ISO 8601 basic time notation, no time zone 383
  B8601TZ format, ISO 8601 basic time notation with time zone 169
B8601TZ informat, ISO 8601 basic time notation with time zone 384
E8601TM format, ISO 8601 extended notation, no time zone 180
E8601TM informat, ISO 8601 extended time notation, no time zone 304
E8601TZ format, ISO 8601 extended notation with time zone 183
E8601TZ informat, ISO 8601 extended time notation with time zone 305
extracting from informat values 285
HHMMw.d format 75
HOURw.d format 77
ISO 8601 extended local time with UTC offset 179
MMSSw.d format 86
TIMEAMPw.d format 100
TIMEw.d format 98
TODw.d format 102
TIMEAMPw.d format 100
TIMEw. informat 354
TIMEw.d format 98
TODSTAMPw. informat 356
compared to MSECw. informat 321
TODw.d format 102
trailing plus or minus sign 357
TRAILSGNW. informat 357
transcoding 10
TUw. informat 358

U
unsigned integer binary (fixed-point) values
IBM mainframe format 206
unsigned integer binary data, reading
IBM mainframe format 338
unsigned packed decimal data
IBM mainframe format 209
unsigned packed decimal data, reading 330
IBM mainframe format 341
unsigned packed decimal format 200
unsigned zoned decimal data
IBM mainframe format 218
unsigned zoned decimal data, reading 351
IBM mainframe format
uppercase
$UPCASEw. informat 275
converting character data to 45
reading data as 275
writing character data in 144
user-defined formats 7

V
VAXRBw.d format 104
VAXRBw.d informat 359
VMS
zoned numeric data 220, 359
VMS format
real binary (floating-point) data in 104
VMSZNW. format 220
VMSZNW. informat 359

W
w.d format 105
w.d informat 361, 369
compared to NUMw.d informat 322
compared to ZDw.d informat 367
WEEKDATEw. format 106
WEEKDATXw. format 107
WEEKDAYw. format 109
weeks
number of week, date value, U algorithm 394
week number, date value, V algorithm 396
week number, date value, W algorithm 398
week number, decimal format, U algorithm 110
week number, decimal format, V algorithm 111
week number, decimal format, W algorithm 113
WEEKU 110
WEEKUw. format 110
WEEKUw. informat 394
WEEKVw. format 111
WEEKVw. informat 396
WEEKWw. format 113
WEEKWw. informat 398
WORDDATEw. format 221
WORDDATXw. format 223
WORDFw. format 224
words
writing numeric values as 225
WORDSw. format 225
writing character data 39, 46

Y
YEARw. format 115
YMDDTMw.d finformat 400
Yymmmddw. format 117
Yymmmddw. informat 362
Yyymddxw. format 119
Yyymmnw. informat 364
Yyymnw. format 116
YYMMxw. format 226
YYMONw. format 121
YYQRw. format 125
YYQRxw. format 126
YYQw. format 122
YYQw. informat 365
YYQxw. format 123

Z
ZDBw.d informat 368
ZDVw.d informat 369
See also w.d informat
See also ZDw.d informat
compared to ZDw.d informat 367
ZDw.d format 227
ZDw.d informat 367, 369
compared to ZDVw.d 370
zero
numeric data with leading zeros 128
zeros, binary
converting to blanks 265
zoned decimal data 11, 240
defined 239
definition 10
formats and 10
formats and informs for 242
IBM mainframe format 213
languages supporting 12, 241
platforms supporting 11, 241
summary of formats and informs 12
zoned decimal data, reading 367, 368
IBM mainframe format 345
zoned decimal format 227
zoned decimal leading-sign data
IBM mainframe format 214
zoned decimal leading-sign data, reading IBM mainframe format 347
zoned decimal separate leading-sign data IBM mainframe format 215
zoned decimal separate trailing-sign data IBM mainframe format 216
zoned decimal separate trailing-sign data, reading IBM mainframe format 350
zoned numeric data
MicroFocus COBOL 220, 359
VMS 220, 359
zoned separate leading-sign data, reading IBM mainframe format 349
Zw.d format 128
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.