CHAPTER

16

Formats

SAS Formats under Windows

A SAS format is an instruction or template that the SAS System uses to write data values. Most SAS formats are described completely in *SAS Language Reference: Dictionary*. The formats that are described here have behavior that is specific to Windows.

Many of the SAS formats that have details specific to the Windows operating environment are used to write binary data. In using these formats, it is important that you understand the concepts that are presented in "Writing Binary Data" on page 317.

If you have formats that you created for use in earlier releases of the SAS System, see "Converting User-Written Formats from Earlier Releases to Version 8" on page 318 for information about how to convert those formats for use with Version 8.

Writing Binary Data

Different computers store numeric binary data in different forms. IBM 370, Hewlett-Packard 9000, Data General ECLIPSE, and Prime computers store bytes in one order. Microcomputers that are compatible with IBM microcomputers and some computers manufactured by Digital Equipment Corporation store bytes in a different order called *byte-reversed*.

Binary data stored in one order cannot be read by a computer that stores binary data in the other order. When you are designing SAS applications, try to anticipate how your data will be read and choose your formats and informats accordingly.

The SAS System provides two sets of informats for reading binary data and corresponding formats for writing binary data.

- The IB*w.d*, PD*w.d*, PIB*w.d*, and RB*w.d* informats and formats read and write in native mode, that is, using the byte-ordering system that is standard for the machine.
- The S370FIB*w.d*, S370FPD*w.d*, S370FRB*w.d*, and S370FPIB*w.d* informats and formats read and write according to the IBM 370 standard, regardless of the native mode of the machine. These informats and formats allow you to write SAS programs that can be run in any SAS environment, regardless of how numeric data are stored.

If a SAS program that reads and writes binary data runs on only one type of machine, you can use the native mode informats and formats. However, if you want to write SAS programs that can be run on multiple machines using different byte-storage systems, use the IBM 370 formats and informats. The purpose of the IBM 370 informats and formats is to enable you to write SAS programs that can be run in any SAS environment, no matter what standard you use for storing numeric data.

For example, suppose you have a program that writes data with the PIB*w.d* format. You execute the program on a microcomputer so that the data are stored in byte-reversed mode. Then on the microcomputer you run another SAS program that uses the PIB*w.d* informat to read the data. The data are read correctly because both of the programs are run on the microcomputer using byte-reversed mode. However, you cannot upload the data to a Hewlett-Packard 9000-series machine and read the data correctly because they are stored in a form native to the microcomputer but foreign to the Hewlett-Packard 9000. To avoid this problem, use the S370FPIB*w.d* format to write the data; even on the microcomputer, this causes the data to be stored in IBM 370 mode. Then read the data using the S370FPIB*w.d* informat. Regardless of what type of machine you use when reading the data, they are read correctly.

Converting User-Written Formats from Earlier Releases to Version 8

You must convert Release 6.04, Release 6.06, and Release 6.08 user-written formats to their Version 8 counterparts before you can use them in a Version 8 SAS program. The only exception to this rule is user-written informats and formats created by Release 6.08 or later under Windows; these informats and formats can be read directly from your Windows SAS session. *

Converting Version 6 User-Written Formats

You can convert Release 6.04, 6.06, and 6.08 SAS catalogs that contain user-written informats and formats using one of the following methods:

Converting Release 6.04 catalogs
: use the CNTLOUT= option in the PROC FORMAT statement in Release 6.04 to create an output data set, and then use the CNTLIN= option in the PROC FORMAT statement in Version 8 to create the Version 8 informats or formats. You must use the V604 engine in your Version 8 SAS session to read the data set. This method also works for converting from Release 6.06 or 6.08.

Converting Release 6.06 or Release 6.08 catalogs
: use the CPORT and CIMPORT procedures to convert the informats and formats. For more information on the CPORT and CIMPORT procedures, see *SAS*

* However, it is recommended that you use PROC CPORT and PROC CIMPORT to convert older Windows catalogs containing user-written informats and formats to Version 8 if you no longer need to use them in previous releases.

Procedures Guide. This method works for converting from Release 6.06 or Release 6.08 only; it does not work for converting from Release 6.04.

Converting Version 5 User-Written Formats

You must also convert Version 5 user-written formats to their Version 8 counterparts before you can use them in a Version 8 SAS program. (This implies that you are not only converting these files, but you are also transferring them from a remote operating system to your PC). You can convert them using one of the following methods:

- Use the V5TOV6 procedure on the remote operating system to convert the informats and formats to Version 6 format. This implies that the remote operating system has access to Version 6 SAS software. Then, transport the converted informats and formats (as binary files) to your Windows operating environment and use the CIMPORT procedure to complete the conversion.

  Note: The V5TOV6 procedure is not available in Version 8 of the SAS System. You must use this procedure in Release 6 of the SAS System. △
- Use the SUGI supplemental procedure FMTLIB under Version 5 on the remote operating system to create an output data set, transport that data set to your PC, and then use the CNTLIN= option in the PROC FORMAT statement in Version 8 to create the Version 8 formats.

HEX*w*.

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

Category numeric
Width range: 1–16
Default width: 8
Alignment: left
Windows specifics: native floating–point representation

Syntax

HEX*w*.

w
specifies the width of the output field. When you specify a *w* value of 1 through 15, the real binary number is truncated to a fixed-point integer before being converted to hexadecimal notation. When you specify 16 for the *w* value, the floating-point value of the number is used; in other words, the number is not truncated.

See Also

- Formats: HEX*w*. in *SAS Language Reference: Dictionary* and "$HEX*w*." on page 320
- Informat: "HEX*w*." on page 347

$HEX*w*.

Converts character values to hexadecimal values

Category character
Width range: 1–32767
Default width: 4
Alignment: left
Windows specifics: ASCII character–encoding system

Syntax

$HEX*w*.

w
specifies the width of the output field.

Details

The $HEX*w*. format is like the HEX*w*. format in that it converts a character value to hexadecimal notation, with each byte requiring two columns. Under Windows, the $HEX*w*. format produces hexadecimal representations of ASCII codes for characters.

See Also

- Formats: $HEX*w*. in *SAS Language Reference: Dictionary* and "HEX*w*." on page 319
- Informat: "$HEX*w*." on page 348

IB*w.d*

Writes integer binary (fixed-point) numbers

Category numeric
Width range: 1–8
Default width: 4
Decimal range: 0–10
Alignment: left
Windows specifics: native floating-point representation

Syntax

IB*w.d*

w
specifies the width of the output field in bytes (not digits).

d
optionally specifies a scaling factor. When you specify a *d* value, the IB*w.d* format multiplies the number by 10^d, and then applies the integer binary format to that value.

Details

The IB*w.d* format converts a double-precision number and writes it as an integer binary (fixed-point) value. Integers are stored in integer-binary (fixed-point) form.

For more information about microcomputer fixed-point values, see Intel Corporation's *i486 Microprocessor Programmer's Reference Manual*.

Examples

Example 1: Processing a Positive Number If you format 1.0 as the double-precision number, it is stored as an integer:

```
01 00 00 00 00 00 00 00
```

(Remember, Windows stores binary data in byte-reversed order.) The value written depends on the *w* value you specify.

If you specify the IB4. format, you receive the following value:

```
01 00 00 00
```

If you specify the IB2. format, you receive the following value:

```
01 00
```

Example 2: Processing a Negative Number If you try to format –1 with the IB4. format, you receive the following value:

```
FF FF FF FF
```

If you specify the IB2. format, you receive the following value:

```
FF FF
```

See Also

- Format: IB*w.d* in *SAS Language Reference: Dictionary*
- Informat: "IB*w.d*" on page 348
- "Writing Binary Data" on page 317

PD*w.d*

Writes packed decimal data

Category numeric
Width range: 1–16
Default width: 1
Decimal range: 1–31
Alignment: left
Windows specifics: How the values are interpreted as negative or positive

Syntax

PD*w.d*

w
specifies the width of the output field in bytes (not digits).

d
optionally specifies a scaling factor. When you specify a *d* value, the PD*w.d* format multiplies the number by 10^d, and then applies the packed decimal format to that value.

Details

The PD*w.d* format writes double-precision numbers in packed decimal format. In packed decimal data, each byte contains two digits. The *w* value represents the number of bytes, not the number of digits. The value's sign is in the uppermost bit of the first byte (although the entire first byte is used for the sign).

Examples

Example 1: Processing a Positive Number If you format 1143.0 using the PD2. format, you receive the following value:

```
00 43
```

If you specify PD4., you receive the following value:

```
00 00 11 43
```

Example 2: Processing a Negative Number If you format –1143.0 using the PD2. format, you receive the following value:

```
80 43
```

If you specify the PD4. format, you receive the following value:

```
80 00 11 43
```

See Also

- Format: PD*w.d* in *SAS Language Reference: Dictionary*
- Informat: "PD*w.d*" on page 350
- "Writing Binary Data" on page 317

PIB*w.d*

Writes positive integer binary data

Category numeric
Width range: 1–8
Default width: 1
Decimal range: 0–10
Alignment: left
Windows specifics: native byte-swapped integers

Syntax

PIB*w.d*

w
specifies the width of the output field in bytes (not digits).

d
optionally specifies a scaling factor. When you specify a *d* value, the PIB*w.d* format multiplies the number by 10^d, and then applies the positive integer binary format to that value.

Details

The PIB*w.d* format converts a fixed-point value to an integer binary value. If the fixed-point value is negative, the PIB*w.d* format writes the integer representation for –1.

For more information about microcomputer fixed-point values, see Intel Corporation's *i486 Microprocessor Programmer's Reference Manual*.

See Also

- Format: PIB*w.d* in *SAS Language Reference: Dictionary*
- Informat: "PIB*w.d*" on page 351
- "Writing Binary Data" on page 317

RB*w.d*

Writes real binary (floating-point) data

Category numeric
Width range: 2–8
Default width: 4
Decimal range: 0–10
Alignment: left
Windows specifics: native floating–point representation

Syntax

RB*w.d*

w
specifies the width of the output field.

d
optionally specifies a scaling factor. When you specify a *d* value, the RB*w.d* format multiplies the number by 10^d, and then applies the real binary format to that value.

Details

The RB*w.d* format writes numeric data in real binary (floating-point) notation. Numeric data for scientific calculations are commonly represented in floating-point notation. (The SAS System stores all numeric values in floating-point notation.) A floating-point value consists of two parts: a mantissa that gives the value and an exponent that gives the value's magnitude.

Real binary is the most efficient format for representing numeric values because the SAS System already represents numbers this way and no conversion is needed.

For more information about Windows floating-point notation, see Intel Corporation's *i486 Microprocessor Programmer's Reference Manual*.

See Also

- Format: RB*w.d* in *SAS Language Reference: Dictionary*
- Informat: "RB*w.d*" on page 352
- "Writing Binary Data" on page 317

ZD*w.d*

Writes zoned decimal data

Category numeric
Width range: 1–32
Default width: 1
Decimal range: 1–10
Alignment: left
Windows specifics: Last byte includes the sign.

Syntax

ZD*w.d*

w
specifies the number of bytes (not the number of digits).

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

Details

The ZD*w.d* format writes zoned decimal data. This is also known as an overprint trailing numeric format. In the Windows operating environment, the last byte of the field contains the sign information of the number. The following table gives the conversion for the last byte.

Digit	ASCII Character	Digit	ASCII Character
0	{	–0	}
1	A	–1	J
2	B	–2	K
3	C	–3	L
4	D	–4	M
5	E	–5	N
6	F	–6	O
7	G	–7	P
8	H	–8	Q
9	I	–9	R

See Also

- Format: ZD*w.d* in *SAS Language Reference: Dictionary*
- Informat: “ZD*w.d*” on page 353

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS Companion for the Microsoft Windows Environment, Version 8,* Cary, NC: SAS Institute Inc., 1999. pp.555.

SAS Companion for the Microsoft Windows Environment, Version 8

ISBN 1–58025–524–8

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

1st printing, September 1999

The Institute is a private company devoted to the support and further development of its software and related services.