CHAPTER

13

Formats

SAS Formats under OpenVMS

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 the SAS System under OpenVMS.

Many of the SAS formats that have details that are specific to the OpenVMS operating environment are used to write binary data. For more information, see "Writing Binary Data" on page 263.

Writing Binary Data

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

Binary data that are 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 you run another SAS program on the microcomputer 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.

HEX*w*.

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

Language element: format

Category: numeric

Width range: 1 to 16

Default width: 8

Alignment: left

OpenVMS specifics: ASCII character-encoding system

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.

Details Each byte requires two columns to represent the corresponding hexadecimal digits. Under OpenVMS, the hexadecimal format of the number is stored in ASCII representation. For example, the decimal integer 17 has a hexadecimal value of 11, so the HEX2. format of 17 is 11.

If HEX16. is specified, the floating-point number is not converted to an integer, and you receive the hexadecimal representation of the native floating-point representation. The bytes of the number are printed so that the left-most byte is of the lowest significance. For more information about OpenVMS floating-point representation, see *Architecture Reference Manual for Alpha* and *Architecture Reference Manual for VAX*.

See Also

- Formats: HEX*w*. in *SAS Language Reference: Dictionary* and "$HEX*w*." on page 265
- Informat: "HEX*w*." on page 322

$HEX*w*.

Converts character values to hexadecimal values

Language element: format
Category: character
Width range: 1 to 32767
Default width: 4
Alignment: left
OpenVMS 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 OpenVMS, the $HEX*w*. format produces hexadecimal representations of ASCII codes for characters.

See Also

- Formats: $HEX*w*. in *SAS Language Reference: Dictionary* and "HEX*w*." on page 264
- Informat: "$HEX*w*." on page 323

IB*w.d*

Writes numbers in integer binary (fixed-point) format

Language element: format
Category: numeric
Width range: 1 to 8
Default width: 4
Decimal range: 0 to 10

Alignment: left

OpenVMS specifics: twos-complement notation; overflow behavior

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 the 10^{d} value, then applies the integer binary format to that value.

Details Negative values are stored in twos-complement notation. If a noninteger value is formatted, rounding occurs. If the value to be formatted is too large to fit in a field of the specified width, then the IB*w.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.

For more information about OpenVMS native fixed-point values, see *Architecture Reference Manual for Alpha* and *Architecture Reference Manual for VAX*.

Example

If you format the value 300 using the IB1. format, you receive the following value:

```
127
```

which is the largest positive value that fits in a byte.

If you format the value -300 using the IB1. format, you receive the following value:

```
-128
```

See Also

- Format: IB*w.d* in *SAS Language Reference: Dictionary*
- Informat: “IB*w.d*” on page 323
- “Writing Binary Data” on page 263

PD*w.d*

Writes values in packed decimal format

Language element: format

Category: numeric

Width range: 1 to 16
Default width: 1
Decimal range: 0 to 10
Alignment: left
OpenVMS specifics: overflow behavior

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 the 10^{d} value, then applies the packed decimal format to that value.

Details Under OpenVMS, if the value to be formatted is too large to fit in a field of the specified width, then the PD*w.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

If you format the value 300 using the PD1. format, you receive the following hexadecimal string:

```
'9C'x
```

which is the packed decimal representation for 9.

If you format the value -300 using the PD1. format, you receive the following hexadecimal string:

```
'9D'x
```

which is the packed decimal representation for -9.

See Also

- Format: PD*w.d* in *SAS Language Reference: Dictionary*
- Informat: “PD*w.d*” on page 324
- “Writing Binary Data” on page 263

PIB*w.d*

Writes positive integer-binary fixed-point values

Language element: format
Category: numeric
Width range: 1 to 8
Default width: 1
Decimal range: 0 to 10
Alignment: left
OpenVMS specifics: overflow behavior

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 the 10^d value, then applies the positive integer binary format to that value.

Details If the value to be formatted is too large to fit in a field of the specified width, then this 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.

If a noninteger value is formatted, rounding occurs.

For more information about OpenVMS native fixed-point values, see *Architecture Reference Manual for Alpha* and *Architecture Reference Manual for VAX*.

Example

If you format the value 300 with the PIB1. format, you receive the following value:

```
255
```

which is the largest unsigned value that fits in a 1-byte field.

See Also

- Format: PIB*w.d* in *SAS Language Reference: Dictionary*
- Informat: “PIB*w.d*” on page 325
- “Writing Binary Data” on page 263

RB*w.d*

Writes numeric data in real-binary (floating-point) notation

Language element: format
Category: numeric
Width range: 2 to 8
Default width: 4
Decimal range: 0 to 10
Alignment: left
OpenVMS 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 the 10^{d} value, then applies the real binary format to that value.

Details Under OpenVMS, the RB*w.d* format causes floating-point numbers to be formatted in the native floating-point representation. 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 OpenVMS floating-point representation, see *Architecture Reference Manual for Alpha* and *Architecture Reference Manual for VAX*.

See Also

- Format: RB*w.d* in *SAS Language Reference: Dictionary*
- Informat: "RB*w.d*" on page 326
- "Writing Binary Data" on page 263

UIC*w.*

Converts a SAS numeric value to an OpenVMS UIC string

Language element: format
Category: numeric
Width range: 16 to 31
Default width: 31
Alignment: left

OpenVMS specifics: All aspects are host-specific

Syntax

UIC*w*.

w
is the width of the output field and must be 16 through 31 (numbers less than 16 or greater than 31 are invalid). If no *w* value is specified, then UIC defaults to the width of the resulting output string.

Details

The UIC value is a 32-bit value, where the upper 16 bits is the group value and the lower 16 bits is the member value. The UIC values that are displayed as two numbers are octal numbers. If alphanumeric values exist for the numbers, the alphanumerics will be displayed. For example:

```
data_null_;
   x=4194377;
   put x uic.;
   run;
```

produces

```
[HOSTVAX,MYIDENT]
```

A system manager assigns a user identification code (UIC) to each OpenVMS user and stores the UIC in the user authorization file (UAF). Each UIC consists of a member identifier and (optionally) a group identifier, enclosed in square brackets, as follows:

[<*group-identifier*,>*member-identifier*]

where *member-identifier* and *group-identifier* can be either names, numbers, or hexadecimal representations.

See Also

□ Function: “GETQUOTA” on page 298

VMSMSG*w*.

Writes numeric values as character strings that contain the equivalent OpenVMS message

Language element: format
Category: numeric
Width range: 16 to 200, where *w* is 16 to 20
Alignment: left
OpenVMS specifics: All aspects are host-specific

Syntax

VMSMSG*w*.

w
specifies the width of the output field.

Details Data formatted using the VMSMSG*w*. format are ASCII strings. Symbolic FAO (Formatted ASCII Output) substitution is not performed.

Example

If you format the value 1 using the VMSMSG. format in the following SAS statement:

```
put rc vmsmsg.;
```

the result is %SYSTEM-S-NORMAL, which is an ASCII string indicating normal successful completion.

VMSTIMEF.

Converts a SAS date-time value to an 8-byte binary value in OpenVMS date and time format

Language element: format
Category: date and time
Width range: 8
Default width: 8
Alignment: left
OpenVMS specifics: All aspects are host-specific

Syntax

VMSTIMEF.

Details The VMSTIMEF. format is specific to OpenVMS. You cannot specify a width with this format; the width is always 8 bytes.

OpenVMS date and time values that are read in with the VMSTIME. informat retain precision up to 1/100 of a second, even though the SAS System cannot display anything less than whole seconds. If you later use the VMSTIMEF. format to write out the date-time value, the precision is retained.

See Also

- Informat: “VMSTIME.” on page 328

VMSZN*w.d*

Generates a string of ASCII digits

Language element: format

Category: numeric

Width range: 1 to 32

Default width: 1

Alignment: left

OpenVMS specifics: All aspects are host-specific

Syntax

VMSZN*w.d*

w
specifies the width of the output field

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

Details The VMSZN*w.d* format is similar to the ZD*w.d* format. Both generate a string of ASCII digits, and the last digit is a special character that denotes the magnitude of the last digit and the sign of the entire number. The difference between these formats is in the special character that is used for the last digit. The following table shows the special characters that are used by the VMSZN*w.d* format.

Desired Digit	Special Character	Desired Digit	Special Character
0	0	–0	p
1	1	-1	q
2	2	-2	r
3	3	-3	s
4	4	-4	t
5	5	-5	u
6	6	-6	v
7	7	-7	w
8	8	-8	x
9	9	-9	y

Data formatted using the VMSZN*w.d* format are ASCII strings.

If the value to be formatted is too large to fit in a field of the specified width, then the VMSZN*w.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.

Examples

Example 1: Using the VMSZN*w.d* Format in a SAS Statement If you format the value 1234 using the VMSZN*w.d* format in the following SAS statement:

```
put i vmszn4.;
```

the result is 1234, which is an ASCII string.

If you format the value 1234 using the VMSZN*w.d* format in the following SAS statement:

```
put i vmszn5.1;
```

the result is 12340, which is an ASCII string.

If you format the value 1234 using the VMSZN*w.d* format in the following SAS statement:

```
put i vmszn6.2;
```

the result is 123400, which is an ASCII string.

If you format the value -1234 using the VMSZN*w.d* format in the following SAS statement:

```
put i vmszn5.;
```

the result is 123t, which is an ASCII string.

Example 2: Zoned-Numeric Representations If you format the value 300 using the VMSZN*w.d* format, you receive the following value:

```
'9'
```

which is an ASCII string.

If you format the value -300 using the VMSZN*w.d* format, you receive the following value:

```
'y'
```

which is an ASCII string that is the zoned-numeric representation of -9.

See Also

- Format: ZD*w.d* in *SAS Language Reference: Dictionary*
- Informats: "VMSZN*w.d*" on page 328 and "ZD*w.d*" on page 330

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Companion for the OpenVMS Environment, Version 8,* Cary, NC: SAS Institute Inc., 1999. 518 pp.

SAS® Companion for the OpenVMS Environment, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

1–58025–526–4

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, by any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

U.S. Government Restricted Rights Notice. Use, duplication, or disclosure of the software by the government is subject to restrictions as set forth in FAR 52.227–19 Commercial Computer Software-Restricted Rights (June 1987).

1st printing, October 1999

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

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

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