
Introduction to SAS Informats and Formats

1.1 Chapter Overview.....	2
1.2 Using SAS Informats	2
1.2.1 INPUT Statement	3
1.2.2 INPUT Function	7
1.2.3 INPUTN and INPUTC Functions.....	8
1.2.4 ATTRIB and INFORMAT Statements	8
1.3 Using SAS Formats.....	9
1.3.1 FORMAT Statement in Procedures	10
1.3.2 PUT Statement.....	11
1.3.3 PUT Function.....	13
1.3.4 PUTN and PUTC Functions.....	14
1.3.5 BESTw. Format.....	14
1.4 Additional Comments	17

1.1 Chapter Overview

In this chapter we will review how to use SAS informats and formats. We will first review a number of internal informats and formats that SAS provides, and discuss how these are used to read data into SAS and format output. Some of the examples will point out pitfalls to watch for when reading and formatting data.

1.2 Using SAS Informats

Informats are typically used to read or input data from external files called flat files (text files, ASCII files, or sequential files). The informat instructs SAS on how to read data into SAS variables. SAS informats are typically grouped into three categories: character, numeric, and date/time. Informats are named according to the following syntax structure:

Character Informats: **\$INFORMAT***w*.
Numeric Informats: **INFORMAT***w.d*
Date/Time Informats: **INFORMAT***w*.

The \$ indicates a character informat. **INFORMAT** refers to the sometimes optional SAS informat name. The *w* indicates the width (bytes or number of columns) of the variable. The *d* is used for numeric data to specify the number of digits to the right of the decimal place. All informats must contain a decimal point (.) so that SAS can differentiate an informat from a SAS variable.

SAS 9 lists other informat categories besides the three mentioned. Some of these are for reading Asian characters and Hebrew characters. The reader is left to explore these other categories.

SAS provides a large number of informats. The complete list is available in SAS Help and Documentation. In this text, we will review some of the more common informats and how to use them. Check SAS documentation for specifics on reading unusual data.

1.2.1 INPUT Statement

One use of SAS informats is in DATA step code in conjunction with the INPUT statement to read data into SAS variables. The first example we will look at will read a hypothetical data file that contains credit card transaction data. Each record lists a separate transaction with three variables: an ID (account identifier), a transaction date, and a transaction amount. The file looks like this:

ID	Transaction Date	Transaction Amount
124325	08/10/2003	1250.03
7	08/11/2003	12500.02
114565	08/11/2003	5.11

The following program is used to read the data into a SAS data set. Since variables are in fixed starting columns, we can use the column-delimited INPUT statement.

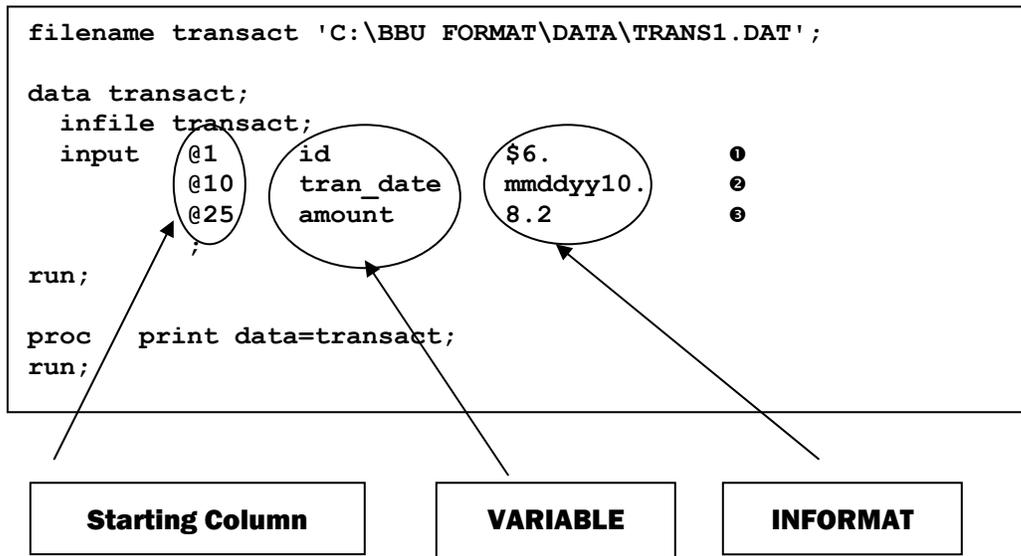


Figure 1.1

The ID variable is read in as a character variable using the **\$6.** informat in line ①. The **\$w.** informat tells SAS that the variable is character with a length **w**. The **\$w.** informat will also left-justify the variable (leading blanks eliminated). Later in this section we will compare results using the **\$CHARw.** informat, which retains leading blanks.

Line ② instructs SAS to read in the transaction date (Tran_Date) using the date informat **MMDDYYw.** Since each date field occupies 10 spaces, the **w.** qualifier is set to 10.

Line ③ uses the numeric informat **8.2.** The **w.d** informat provides instruction to read the numeric data having a total width of 8 (8 columns) with two digits to the right of the decimal point. SAS will insert a decimal point only if it does not encounter a decimal point in the specified **w** columns. Therefore, we could have coded the informat as **8.** or **8.2.**

The PROC PRINT output is shown here. Note that the Tran_Date variable is now in terms of SAS date values representing the number of days since the first day of the year specified in the YEARCUTOFF option (for this run, yearcutoff=1920).

Obs	id	tran_date	amount
1	124325	15927	1250.03
2	7	15928	12500.02
3	114565	15928	5.11

Output 1.1

We can make this example a bit more complicated to illustrate some potential problems that typically arise when reading from flat files. What if the Amount variable contained embedded commas and dollar signs? How would we generate

the code to read in these records? Here is the modified data with the code that reads the file using the correct informat instruction:

124325	08/10/2003	\$1,250.03
7	08/11/2003	\$12,500.02
114565	08/11/2003	5.11

```
filename transact 'C:\BBU FORMAT\DATA\TRANS1.DAT';

data transact;
  infile transact;
  input @1 id          $6.
        @10 tran_date mmdyy10.
        @25 amount    comma10.2    ❶
        ;
run;

proc print data=transact;
run;
```

Line ❶ uses the numeric informat named **COMMAw.d** to tell SAS to treat the Amount variable as numeric and to strip out leading dollar signs and embedded comma separators. The PROC PRINT output is shown here:

Obs	id	tran_date	amount
1	124325	15927	1250.03
2	7	15928	12500.02
3	114565	15928	5.11

Output 1.2

Note that the output is identical to the previous run when the data was not embedded with commas and dollar signs. Also note that the width of the informat in the code is now larger (10 as opposed to 8 to account for the extra width taken up by commas and the dollar sign). What seemed like a programming headache was solved simply

by using the correct SAS informat. When you come across nonstandard data, always check the documented informats that SAS provides.

Now compare what would happen if we changed the informat for the ID variable from a **\$w.** informat to a **\$CHARw.** informat. Note that the **\$CHARw.** informat will store the variable with leading blanks.

```
filename transact 'C:\BBU FORMAT\DATA\TRANS1.DAT';

data transact;
  infile transact;
  input @1 id          $CHAR6.
        @10 tran_date mmddy10.
        @25 amount    comma10.2
        ;
run;

proc print data=transact;
run;
```

Obs	id	tran_date	amount
1	124325	15927	1250.03
2	7	15928	12500.02
3	114565	15928	5.11

Output 1.3

Note that the ID variable now retains leading blanks and is right-justified in the output.

1.2.2 INPUT Function

You can use informats in an INPUT function within a DATA step. As an example, we can convert the ID variable used in the previous example from a character variable to a numeric variable in a subsequent DATA step. The code is shown here:

```
data transact2;
  set transact;
  id_num = input(id,6.);
proc print data=transact2;
run;
```

The INPUT function in line ① returns the numeric variable Id_Num. The line states that the ID variable is six columns wide and assigns the numeric variable, Id_Num, by using the numeric **w.d** informat. Note that when using the INPUT function, we do not have to specify the *d* component if the character variable contains embedded decimal values. The output of PROC PRINT is shown here. Note that the Id_Num is right-justified as numeric values should be.

Obs	id	tran_ date	amount	id_num
1	124325	15927	1250.03	124325
2	7	15928	12500.02	7
3	114565	15928	5.11	114565

Output 1.4

Also note that the resulting informat for the variable assigned using the INPUT function is set to the type of informat used in the argument. In the above example, since **6.** is a numeric informat, the Id_Num variable will be numeric.

1.2.3 INPUTN and INPUTC Functions

The INPUTN and INPUTC functions allow you to specify numeric or character informats at run time. A modified example from SAS 9 Help and Documentation shows how to use the INPUTN function to switch informats that are dependent on values of another variable.

```
options yearcutoff=1920;

data fixdates (drop=start readdate);
  length jobdesc $12 readdate $8;
  input source id lname $ jobdesc $ start $;
  if source=1 then readdate= 'date7.  ';
  else readdate= 'mmddy8.';
  newdate = inputn(start, readdate);
  datalines;
  1 1604 Ziminski writer 09aug90
  1 2010 Clavell editor 26jan95
  2 1833 Rivera writer 10/25/92
  2 2222 Barnes proofreader 3/26/98
  ;
```

Note that the INPUTC function works like the INPUTN function but uses character informats. Also note that dates are numeric, even though we use special date informats to read the values.

1.2.4 ATTRIB and INFORMAT Statements

The ATTRIB statement can assign the informat in a DATA step. Here is an example of the DATA step in Section 1.2.1 rewritten using the ATTRIB statement:

```
data transact;
  infile transact;
  attrib id          informat=$6.
         tran_date  informat=mmddy10.
         amount     informat=comma10.2
  ;
```

```

input @1 id
      @10 tran_date
      @25 amount
      ;
run;

```

This next example shows how we could also use the INFORMAT statement to read in the data as well. With SAS there is always more than one way to get the job done.

```

data transact;
  infile transact;
  informat id          $6.
           tran_date  mmddyy10.
           amount     comma10.2
           ;
  input @1 id
        @10 tran_date
        @25 amount
        ;
run;

```

1.3 Using SAS Formats

If informats are instructions for reading data, then you can view formats as instructions for outputting data. Using the data provided above, we will review how to use some formats that SAS provides.

Since formats are primarily used to format output, we will look at how we can use existing SAS internal formats using the FORMAT statement in PROCs.

1.3.1 FORMAT Statement in Procedures

Return to the first example introduced in Section 1.2.1 and modify PROC PRINT to include a FORMAT statement that would return dates in standard mm/dd/yyyy format and list transaction amounts using dollar signs and commas. Here is the code:

```
options center;
filename transact 'C:\BBU FORMAT\DATA\TRANS1.DAT';

data transact;
  infile transact;
  input @1 id          $6.
        @10 tran_date mmddy10.
        @25 amount    8.2
        ;
run;

proc print data=transact;
  format tran_date mmddy10.
         amount    dollar10.2;
run;
```



FORMAT STATEMENT IN PROC

Obs	id	tran_date	amount
1	124325	08/10/2003	\$1,250.03
2	7	08/11/2003	\$12,500.02
3	114565	08/11/2003	\$5.11

Output 1.5

Notice that we used a **DOLLARw.d** format to write out the Amount variable with a dollar sign and comma separators. If we used a **COMMAw.d** format, the results would be similar but without the dollar sign. We see that the **COMMAw.d** informat used in Section 1.2.1 has a different function from the **COMMAw.d** format. The informat ignores dollar signs and commas while the **COMMAw.d** format outputs data with embedded commas without the dollar sign. Check SAS Help and Documentation when using informats and formats since the same-named informat may have a different functionality from the same-named format.

1.3.2 PUT Statement

Informats combined with INPUT statements read in data from flat files. Conversely, we can use formats with the PUT statement to write out flat files. Let's see how to take the Transact SAS data set and write out a new flat file using PUT statements. Recall that the Transact data set was created using the following code:

```
options center;
filename transact 'C:\BBU FORMAT\DATA\TRANS1.DAT';

data transact;
  infile transact;
  input @1 id          $6.
        @10 tran_date mmddyy10.
        @25 amount    8.2
        ;
run;
```

Run the following code to create a new flat file called transact_out.dat:

```
data _null_;
  set transact;
  file 'c:\transact_out.dat';
  put @1 id          $char6.
      @10 tran_date mmddyy10.
      @25 amount    8.2
      ;
run;
```

Some comments about the above code:

- ❶ The data set name `_NULL_` is a special keyword. The `_NULL_` data set does not get saved into the workspace. The keyword turns off all the default automatic output that normally occurs at the end of the DATA step. It is used typically for writing output to reports or files.
- ❷ Use the SET statement to read the transact data into the DATA step.

- ③ Specify the output flat file using the FILE statement. Review SAS documentation for FILE statement options for specific considerations (i.e., specifying record lengths for long files, file delimiters, and/or outputting to other platforms such as spreadsheets).
- ④ Specify the **\$CHARw.** format, but since the ID variable is already left-justified using the **\$w.** informat, the output would be the same if a **\$w.** format had been used.

The data file created from the above code is shown here:

124325	08/10/2003	1250.03
7	08/11/2003	12500.02
114565	08/11/2003	5.11

Output 1.6

If the user of the file requires the ID variable to be right-justified, the following changes to the code can accommodate that request. In this code, a new numeric variable called `id_num` was created, which applies the INPUT function to the character ID variable.

```
data _null_;
  set transact;
  file 'c:\transact_out.dat';
  id_num = input(id,6.);
  put @1 id_num 6.
     @10 tran_date mmdyy10.
     @25 amount 8.2
  ;
run;
```

124325	08/10/2003	1250.03
7	08/11/2003	12500.02
114565	08/11/2003	5.11

What if the user calls back requesting that the ID variable have leading zeros? This is not a problem because SAS has a special numeric format to include leading zeros called **Zw.d**. Here is the modified code and the output file:

```
data _null_;
  set transact;
  file 'c:\transact_out.dat';
  id_num = input(id,6.);
  put @1 id_num      z6.
      @10 tran_date mmddy10.
      @25 amount    8.2
      ;
run;
```

124325	08/10/2003	1250.03
000007	08/11/2003	12500.02
114565	08/11/2003	5.11

The above example is handy to have. Especially if you read zip code data as numeric and then want to output results with leading zeros in flat files, reports, or PROCs.

1.3.3 PUT Function

Like the INPUT function, SAS also has the PUT function to use with SAS variables and formats to return character variables. The format applied to the source variable must be the same type as the source variable—numeric or character.

For example, what if we have a data set with a 13-digit numeric variable called `Accn_Id` and we want to generate a character variable called `Char_Accn_Id` from the numeric variable with leading zeros? The following PUT function can be applied in a DATA step:

```
char_accn_id = put(accn_id,z13.);
```

Note that the PUT function always returns a character variable while the INPUT function returns a type (numeric or character) dependent on the informat used in the argument.

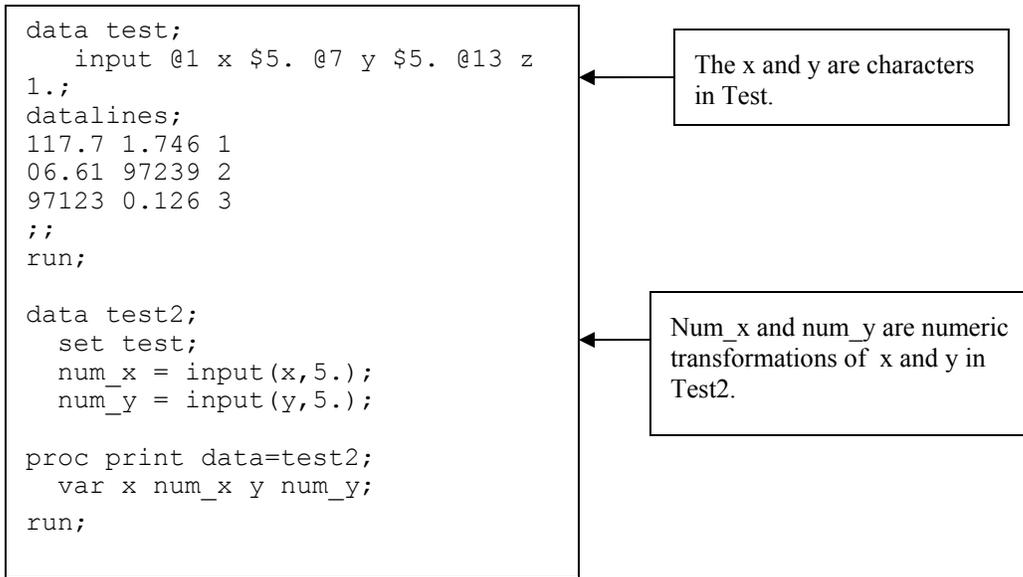
1.3.4 PUTN and PUTC Functions

These functions work like the INPUTN and INPUTC functions reviewed in Section 1.2.2. The functions allow you to name a format during run time. A detailed example is shown in Chapter 5, Section 5.3.

1.3.5 BESTw. Format

When outputting numeric data without a format specification, SAS uses the default **BESTw.** format. You can increase the width of the numeric display by overriding the default **BESTw.** format by explicitly declaring a **BESTw.** format in a format specification.

To make the concept clear, we'll look at the problem of converting character to numeric data that was introduced in Section 1.2.2. The INPUT function can be used to convert character data to numeric. Here is another example and code:

**Figure 1.2**

When we look at the output of the run we get the following results, which at first look strange:

Obs	x	num_x	y	num_y
1	117.7	117.70	1.746	1.75
2	06.61	6.61	97239	97239.00
3	97123	97123.00	0.126	0.13

Output 1.7

It looks like the X variable translated correctly, but when we look at the Y variable we notice that digits got rounded off in the Num_y variable. Before blaming the INPUT function in creation of data set Test2, try to increase the width of the numeric display using a **BESTw.** format. Here are the code and output:

```
proc print data=test2;
  var x num_x y num_y;
  format num_y best10.;
run;
```

Obs	x	num_x	y	num_y
1	117.7	117.70	1.746	1.746
2	06.61	6.61	97239	97239
3	97123	97123.00	0.126	0.126

Output 1.8

With the **BESTw.** format applied, we see that the character-to-numeric translation was done correctly.

We can apply the **BESTw.** format to all numeric variables as shown in the following change of PROC PRINT, which formats all numeric data in the data set with the best10. format:

```
proc print data=test2;
  var x num_x y num_y;
  format numeric best10.;
run;
```

Obs	x	num_x	y	num_y
1	117.7	117.7	1.746	1.746
2	06.61	6.61	97239	97239
3	97123	97123	0.126	0.126

Output 1.9

1.4 Additional Comments

There are a large number of informats and formats supplied by SAS. Be clear about your data and the format of your data. Don't assume anything. Always check your SAS logs for warnings and errors. Be on the lookout for incorrect *w* or *d* specifications. If you don't specify large enough widths, character variables will be truncated and numeric data might be reformatted.

As a review of this chapter, the following table shows the function and usage of informats and formats:

CONCEPT	FUNCTION	USAGE IN A DATA STEP	USAGE IN A PROC
INFORMAT	Input data.	Use with the INPUT, ATTRIB, or INFORMAT statement. Use with the INPUT, INPUTN, or INPUTC function.	INFORMAT statements are rarely used in PROCs. Exceptions are PROCs that are used to input data such as PROC FSEDIT.
FORMAT	Output data or format data in reports.	Use with the PUT, ATTRIB, or FORMAT statement. Use with the PUT, PUTC, or PUTN function.	Use the ATTRIB or FORMAT statement.

More Information

More information about SAS informats and formats can be found in SAS Help and Documentation.

