Paper 235-29

(In)Formats (In)Decently Exposed

Harry Droogendyk, Stratia Consulting Inc., Lynden, ON

ABSTRACT

If you've pulled data into SAS or pushed it out the other end you *have* used SAS informats and formats, really. Didn't hurt a bit did it? There's nothing to be afraid of, formats are really quite helpful little beasts.

If you've been intimidated or confused by SAS formats, come along for a revealing ride as the shadowy side of data formatting is decently exposed. Learn how many data conversion and presentation issues are easily handled, whether the data be dates, times, numbers or strings. Don't avert your eyes, discover the power, efficiency and freedom of SAS formats, you'll never be the same.

INTRODUCTION

If you've been in the programming business for any length of time, you're well aware that your data is not necessarily stored as you enter it or as it is displayed when presented for your viewing. For example, open up an editor on your ASCII machine and type the word SUGI in the first record and save the file. If we display the file in hexadecimal representation, we do not see 'SUGI', rather, the four bytes appear as '53 55 47 49'. However, the hexadecimal representation is really only a more readable form of the binary format a computer ultimately stores and uses. How does an ASCII machine store the word SUGI? Each of the four bytes contains eight bits, a total of 32 bits, 01010011010101010100011101001001 . That's just plain ugly!

Without getting too technical, matters become even more complicated when we consider numbers, especially those stored in a SAS dataset. To enable SAS to store numbers of large magnitude and perform calculations that require many digits of precision to the right of the decimal point, SAS stores all numbers using **floating-point** representation. You may know this format as *scientific notation*. Values are represented as numbers between 0 and 1 times a power of 10. e.g. the number 1 in scientific notation is $.1 \times 10^1$

We can look at this and do the math and come up with an answer of 1. But, just to confuse the matter even further, SAS stores numbers in scientific notation, not in base 10 (like we humans count), but in base 2. In hexadecimal format the numeral 1 appears as 3F F0 00 00 00 00 00 00. Certainly doesn't look like 1 to me! Ahhh, aren't you glad you don't have to work with the internals?

It's evident that *something* happens to our data after we enter it, *something else* before it's displayed to us. The stuff we can easily read is modified by SAS to a format it can understand, massage and store. In order to make the data meaningful to us, SAS must re-format the data for our reading pleasure when pumping it back out. Thankfully we're not limited to SAS' idea of how things ought to be - we can use informats and formats to direct SAS *how* to input and display our data.

This paper is an attempt to introduce you to SAS-supplied formats and informats and how to use them to effectively and efficiently deal with data conversion and presentation issues.

Let's begin with some definitions and examples. An **informat** is an instruction used to read data values into a SAS variable. In addition, if a variable has not yet been defined, SAS uses the informat to determine whether the variable ought to be numeric or character. The informat is also used to determine the length of character variables. A **format** is defined as an instruction that SAS uses to write data values. Formats are used to control the written appearance of data values. Both informats and formats are of the form: **<$> name <w> . <d>**

- $ required prefix for *character* (in)formats
- format up to 7 chars long for character, 8 for numeric, may not **end** in a number! (v9 - 31/32 long)
- w width of value to be read / written, includes commas, decimal places, dollar signs etc…
- **.** a period, *every* format **must have** a period, distinguishes formats from variable names
- d decimal places, optional, *only for numeric (in)formats*

For example: $char20. - 20 byte character format
dollar12.2 - 12 byte numeric format, with 2 decimal places

HAVE YOU EVER USED FORMATS OR INFORMATS?

"Are you kidding?!", you say, "I haven't been using SAS that long, formats are *confusing*!". Well, I can say with certainty you have used formats, even if SAS was doing it behind the scenes on your behalf. Consider the following simple data step and PRINT procedure:

```
data weather;
  input date city $ degrees_celsius ;
cards;
20040117 Montreal  -134.3456
20040204 Toronto     -2.5
20040328 Calgary      7.1
20040413 Ottawa      12.64
20040510 Lynden      17.2
run;
```

```
proc print data=weather;
run;
```

Obs	date	city	degrees_ celsius
1	20040117	Montreal	-134.346
2	20040204	Toronto	-2.500
3	20040328	Calgary	7.100
4	20040413	Ottawa	12.640
5	20040510	Lynden	17.200

Since neither the data step or PRINT procedure specified input or output formats, SAS has used default formats. How did this default behavior affect what SAS did with the weather data?

SAS read the input data and converted them to its internal format. As we discovered in the introduction, the internal representation will not look anything like the characters in our data. Rather, the value will be stored internally in a format that allows SAS to easily store and manipulate the data efficiently. The PRINT procedure takes the internal values and produces output using default format specifications. It seems as though SAS did a pretty good job - the default formatted output generated by the PRINT procedure looks fine, with one exception - we lost the fourth decimal place in Montreal's January temperature. PRINT defaulted to the **best.** format and decided that all we needed was three decimal places. Maybe defaults aren't good enough.

The use of default informats is only possible if the data is what SAS calls " standard numeric or character format." In other words, while numeric data may contain negative signs and decimal points, it may not contain commas or currency signs. Character data cannot contain embedded blanks. There's going to be a time when you must input and report non-standard data. What happens if we allow SAS to default in those cases?

Note the data error in the example below when the default behavior couldn't deal with the special characters in **accum_parking_revenue** . From the output created by the PUT statement, we can see that two variables were created, but **accum_parking_revenue** contains a missing value.

```
data parking;
  input city $ accum_parking_revenue ;
  put city= accum_parking_revenue=;
cards;
Montreal $145,234.72
run;
```

Log Output:

```
NOTE: Invalid data for accum_parking_revenue  in line 11 10-20.
city=Montreal  accum_parking_revenue=.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8-
-
11         Montreal $145,234.72
city=Montreal  accum_parking_revenue=.  _ERROR_=1  _N_=1
NOTE: The data set WORK.PARKING  has 1 observations  and 2 variables.
```

SPECIFYING (IN)FORMATS

INFORMATS

Why couldn't we simply read the non-standard numeric **accum_parking_revenue** data as character data? Doing so will preserve the currency signs and the commas so it'll still look purty on the way out. But… we cannot use character data in calculations! If we even *suspect* that we'll *ever* need the data for *any* type of calculation (the correct answer is YES!!!) , informats must be explicitly specified to properly INPUT the values.

```
data parking;
  input city $ accum_parking_revenue  dollar12.2 ;
cards;
Montreal $145,234.72
Ottawa $221,691.00
Toronto $275,876.54
Lynden $397,112.23
Hamilton $226,432.02
run;
```

As a bonus, consider the space savings in treating **accum_parking_revenue** as a number rather than a character: 8 numeric bytes vs. at least 11 bytes in character mode.

```
##    Variable                 Type    Len    Pos
-------------------------------------------------
2    accum_parking_revenue    Num       8      0
1    city                     Char      8      8
```

While the following PRINT output displays the same *values* as the input data, the **accum_parking_revenue** values are not formatted as we'd expect for currency data. The informats specified on the INPUT statement ensured SAS read the data correctly, but nothing we've done so far has resulted in anything other than default output formats.

```
proc print data=parking;
run;
                      accum_
                     parking_
Obs      city        revenue

 1     Montreal     145234.72
 2     Ottawa       221691.00
 3     Toronto      275876.54
 4     Lynden       397112.23
 5     Hamilton     226432.02
```

FORMATS

Just as it's possible to explicitly specify informats when reading raw data, *temporary or permanent* formats can also be defined to the columns for output. Utilizing the FORMAT statement in the PRINT procedure is an example of a temporary format assignment.

```
proc print data=parking;
  format accum_parking_revenue  dollar12.2; /*  Temporary format  */
run;
```

```
                        accum_
                       parking_
Obs      city           revenue

 1     Montreal      $145,234.72
 2     Ottawa        $221,691.00
 3     Toronto       $275,876.54
 4     Lynden        $397,112.23
 5     Hamilton      $226,432.02
```

PERMANENT (IN)FORMATS

Temporary formats are only in force for the life of the step in which they are defined. For persistent output format definitions, the format must be stored by SAS in the descriptor portion of the data set. Most often, this is done at dataset creation time via the (IN)FORMAT or ATTRIB statements. When informats are defined in this manner, it is not necessary to specify them again on the INPUT statement.

```
data parking;
  attrib   date label = 'Accum Date';
  informat accum_parking_revenue dollar12.2;
  format   accum_parking_revenue dollar12.2;
  label    accum_parking_revenue = 'Accum. Parking Revenue';
  input date city : $12.  accum_parking_revenue ;
cards;
20040117 Montreal   $145,234.72
20040204 Montreal $221,691.00
20040328 Montreal   $375,876.54
20040413 Montreal $597,112.23
20040510 Montreal   $726,432.02
run;
```

Notice the *much* more interesting CONTENTS listing, now showing permanent formats, informats and labels for two fields:

#	Variable	Type	Len	Pos	Format	Informat	Label
1	date	Num	8	0			Accum Date
2	accum_parking_revenue	Num	8	8	DOLLAR12.2	DOLLAR12.2	Accum. Parking Revenue
3	city	Char	12	16			

Since permanent output formats have been defined for the **accum_parking_revenue** column, there' s no need to use the (temporary) FORMAT statement in the PRINT procedure:

```
proc print data=parking label;
run;
```

Obs	Accum Date	Accum. Parking Revenue	city
1	20040117	$145,234.72	Montreal
2	20040204	$221,691.00	Montreal
3	20040328	$375,876.54	Montreal
4	20040413	$597,112.23	Montreal
5	20040510	$726,432.02	Montreal

SAS DATE / TIME VALUES

We've talked about converting data values, particularly numeric items, into SAS' internal format, a format more suited for storage and computation. There's one more very important class of values that must be highlighted: those relating to date and time.

If we were to take the data from the previous example, and calculate the average parking revenue per day, based on the **date** field values, we'd first have to calculate how many days had elapsed between observations. The year, month and day values could be parsed out of the date value and the appropriate arithmetic gymnastics performed to subtract the chunks, paying special attention to number of days / month, year rollovers, leap years etc… Of course there's a better way or it wouldn't be mentioned in a tutorial paper!

SAS has the ability to store dates *in a numeric field* as the number of elapsed days since January 1, 1960. In other words, Jan 2, 1960 has a SAS date value of 1, Dec 31, 1960 is 365, May 10, 2004 is 16,201. If the appropriate informats are used on INPUT, SAS will convert our readable date values to a SAS date value. Once our dates are in SAS date format, the number of days between observations is a simple subtraction between the two date values. In addition, it allows us to use a plethora of date functions and output formats to effectively process and present date values.

In an analogous fashion, SAS time formats store times as the number of seconds since midnight. 1:00 am. is stored as 3600, 1:30 as 5400, 2:00 as 7200 etc… Again, making time calculations much simpler.

Note the **date** informat and output format specifications:

```
data parking;
  attrib   date informat = yymmdd10. format = mmddyyd10. label = 'Accum Date';
  informat accum_parking_revenue      dollar12.2;
  format   accum_parking_revenue
           avg_daily_parking_revenue  dollar12.2;
  label    accum_parking_revenue      = 'Accum. Parking Revenue';
  label    avg_daily_parking_revenue  = 'Avg. Daily Parking Revenue';
  input date city : $12. accum_parking_revenue  ;

 avg_daily_parking_revenue  =
     ( accum_parking_revenue  - lag(accum_parking_revenue))  /
     ( date - lag(date)) ;

cards;
20040117 Montreal $145,234.72
20040204 Montreal $221,691.00
20040328 Montreal $375,876.54
20040413 Montreal $597,112.23
20040510 Montreal $726,432.02
run;
```

Obs	Accum Date	Accum. Parking Revenue	Avg. Daily Parking Revenue	city
1	01-17-2004	$145,234.72	.	Montreal
2	02-04-2004	$221,691.00	$4,247.57	Montreal
3	03-28-2004	$375,876.54	$2,909.16	Montreal
4	04-13-2004	$597,112.23	$13,827.23	Montreal
5	05-10-2004	$726,432.02	$4,789.62	Montreal

ADDITIONAL WAYS TO SPECIFY (IN)FORMATS

Thus far, we've seen a couple methods of applying input and output formats to our data:

- INPUT / PUT statements — temporary
- FORMAT (in data step), INFORMAT, ATTRIB statement — permanent

There are occasions when the data will *already* be in a SAS dataset, in a format not suited for our purposes. Perhaps a date field has been read and stored in its external format, e.g. 20040510, or numeric data must be presented in a different format than the permanent format defines. In those cases, it's often necessary to format the data "on the fly" or create additional variables in the format we require using INPUT and PUT *functions*.

Consider an observation containing the data below.

```
data bad_data;
   date        = '20040510';
   amt         = '$123,456.78';
   time        = '08:34';
   postal_code = 'l0r 1t0';
run;
```

Obs	date	amt	time	postal_ code
1	20040510	$123,456.78	08:34	l0r 1t0

To convert the existing SAS data, we can use the INPUT *function* to manipulate the data. The PUT *statement* displays both, new informatted values and the new formatted values, specifying formats different than the original data.

```
data good_data ( keep = new: );
  set bad_data;
  new_date = input(date,yymmdd8.);
  new_amt  = input(amt,dollar14.2);
  new_time = input(time,time5.);
  new_pc   = input(postal_code,$upcase.);

  put new_date  @30 new_date yymmdds10.  /
      new_amt   @30 new_amt  dollar13.2  /
      new_time  @30 new_time tod12.2 /
      new_pc ;
run;
```

Informated:	Formatted:
16201	2004/05/10
123456.78	$123,456.78
30840	08:34:00.00
LOR 1TO	

INPUT and PUT statements may also be used in PROC SQL sentences to convert / reformat **bad_data** on the fly:

```
proc sql;
  select input(date,yymmdd8.)              as date format=yymmdds10.
        ,input(amt,comma12.2)              as amt  format=dollarx12.2
        ,input(time,time5.)                as time format=hour4.1
        ,put(upcase(postal_code),$20.-r)   as postal_code
    from bad_data
  ;
quit;
```

```
date           amt         time   postal_code
-------------------------------------------------
2004/05/10    $123.456,78    8.6               LOR 1TO
```

POTENTIAL PITFALLS WITH PHORMATS, err… FORMATS

How easy is it to have problems with SAS supplied formats? It's pretty difficult, but there are a few gotchas to watch for.

INCORRECTLY SPECIFIED FORMATS

We've seen the difference between character and numeric formats. Character formats must start with a dollar sign, numeric formats may not, e.g. $12. vs 12.2 What happens if we get confused and attempt to use a numeric format in the place of a character format or vice versa? Not surprising, SAS is quite forgiving. Consider the following:

```
a = 'SUGI';
put a binar32.;
```

```
96      what = 'SUGI';
97      put what binar32.;
                --------
                48
ERROR 48-59: The format $BINAR was not found or could not be loaded.
NOTE: The SAS System stopped processing this step because of errors.
```

There's a couple problems with this PUT statement. First there's no $ at the beginning of the format like we'd expect in conjunction with the character field **what**. In addition, the **binary32.** format was misspelled as **binar32.** But did you notice what SAS did? SAS realized the incorrect *type* of format was specified and added the dollar sign for us. That was a bit of a wasted effort since **$binar.** doesn't exist either, but it's the thought that counts right?

Our data step has a serious problem. The incorrect format has caused a real error and aborted processing. If all we're using is SAS-supplied formats, such an error should be a show-stopper and the data step should abort because

it really is a syntax error. However, if we are really attached to the non-existent **$binar.** format and don't want to take it out and it's our earnest desire that the data step continue, the **NOFMTERR** option will ignore format errors. The NOFMTERR option is more important when using user-created formats, but that's a discussion for another paper.

```
options nofmterr;
```

As the errors in the previous data step proved, the default behaviour is **fmterr**. Let's see what the log produced now that we've specified **nofmterr**.

```
161  options nofmterr;
162  data _null_;
163    what = 'SUGI';
164    put 'Invalid format';
165    put '-------------- ';
166    put what binar32. /;
             --------
             484
NOTE 484-185: Format $BINAR was not found or could not be loaded.

167    put 'Correct format';
168    put '-------------- ';
169    put what binary32.;

Invalid format
--------------
SUGI

Correct format
--------------
01010011010101010100011101001001
```

The format error no longer causes the data step to abort. However, the first PUT statement displays **SUGI**, the value of **what** as it is, unaffected by what we had hoped was a valid format. Under "Correct format" we can see the results we really expected. Notice that SAS quietly substituted **$binary.** for the erroneously specified **binary.** without any gloating at all, not even a NOTE: in the log.

INCORRECT FORMAT WIDTH

Way back in the introduction, when we were talking about the components of a format, we found out that the number before the period was the total allowed width. For example, the format **dollar12.2** allows 12 positions for everything, dollar sign, digits, commas, decimal point and, if we're talking the balances in government books, a negative sign.

```
balance = -1234567.89;
put balance dollar10.2;

214     balance = -1234567.89;
215     put balance dollar10.2;
-1234567.9
NOTE: At least one W.D format was too small for the number to be printed. The decimal may be
      shifted by the "BEST" format.
```

SAS has done the math and concluded you can't put 10lbs of stuff in a 5lb bag. Rather than ensuring **balance** was displayed replete with dollar signs and commas, but missing significant digits, SAS defaulted to the **best.** format and pumped out as many significant digits as it could into the allotted 10 bytes.

Sometimes, despite SAS' **best.** efforts, there just isn't room. In those cases a string of asterisks is output along with the " ain't gonna fit " log message. You've seen this same type of behavior in Microsoft Excel which displays # signs when the cell width is not adequate to contain the value in it.

```
widgets = 123;
put widgets z2.;
```

```
218  data _null_;
219     widgets = 123;
220     put widgets z2.;
221  run;

**
NOTE: At least one W.D format was too small for the number to be printed. The decimal may be
      shifted by the "BEST" format.
```

INFORMAT WITH DECIMALS

Many of the numbers we deal with are not integers. The fact that we have to account for decimals leaves us open for another pitfall. Consider the following data step where different amounts with varied precision are read. We've specified an informat with a maximum length of 7 (including decimal point) and two decimal digits.

```
data all_amts;
   format amt 7.2;
   input amt 7.2;
cards;
123.45
3.7
567.88787
67
11.
run;
```

Obs	amt
1	123.45
2	3.70
3	567.89
4	0.67
5	11.00

From the PRINT output, we can see that our disparate data has been handled quite nicely for the most part. It is readily apparent that SAS took the decimal point supplied in the input data at face value and truncated or added zeroes to flesh out the **7.2** format. But, there's problem with **67**. Note that this line of input data did **not** contain a decimal point. When using a W.D numeric informat, SAS divides the input data by 10^D (i.e. 10 ** D) *unless* the input data contains a decimal point. Be consistent or your data values may end up smaller by magnitudes!

EASY LIVING VIA FORMATS

In addition to allowing us to store data items correctly, convert dates and times to internal format for ease of manipulation and present data in a meaningful way, formats can also deliver us from some of the drudgery in our SAS life. Let's look at a few time savers available via the means of SAS formats.

JUSTIFICATION

Typically, character data are left-justified, numeric data right-justified. There's times when you might want to left-justify a numeric value. Perhaps you're ticked at the chief accountant, so why not left-justify all the columns of numbers on that daily report? A more realistic application, creating labels for report lines. Note how the **line** value of **9** is snuggled up to the hyphen when the -L format modifier is added:

```
data _null_;
  do line = 9 to 10;
     left  = 'Line-' || put(line,2.-L);
     right = 'Line-' || put(line,2.);
     put @2 left= / right=;
  end;
run;
```

```
left=Line-9
right=Line- 9

left=Line-10
right=Line-10
```

RETENTION OF LEADING ZEROES

In most numeric data, the high-order, leading zeroes are not required on output. If we have an amount field with values ranging from 4 to 2000, we wouldn't want the lower values to be displayed with leading zeroes, e.g. 0004. However, there are times when leading zeroes are important. Consider part numbers, invoice and cheque numbers and the like:

```
a = 4;     put a @10 a z4.;
4          0004
```

PRESERVATION OF LEADING SPACES

By default, SAS ignores leading spaces. Well, worse than that, it'll get rid of them if it can. If leading spaces are important, you must explicitly tell SAS to keep 'em via the $CHAR. informat at INPUT time:

```
data a;
   input @1 fruit1 $10.      @11 where1 $10.
         @1 fruit2 $char10.  @11 where2 $char10.;  /*  Read line again  */
cards;
12345678901234567890
     Apple    Orchard
    Orange      Grove
     Grape      Store
run;
```

Note **fruit2** and **where2** retained the leading spaces while **fruit1** and **where1** are left-justified:

```
Obs     fruit1         where1          fruit2         where2

 1      1234567890     1234567890     1234567890     1234567890
 2      Apple          Orchard             Apple       Orchard
 3      Orange         Grove              Orange         Grove
 4      Grape          Store               Grape         Store
```

INTERNATIONAL DATE FORMATS

No longer are SAS programmers restricted to displaying dates only in the English language. With the advent of v8 many other languages are now available, e.g. Dutch, Italian, German etc.. See the **International Date and Datetime Formats** section in SAS OnlineDocs for a complete list of languages available.

To use international date formats, the system option **dflang** must be set to one of the available languages:

```
options dflang=dutch;
```

Standard Format	Standard Output	International Format	International (Dutch) Output
DATE9.	10MAY2004	EURDFDE9.	10mei2004
DATETIME.	10MAY04:10:30:00	EURDFDT.	10mei04:10:30:00
DDMMYY10.	10/05/2004	EURDFDD10.	10-05-2004
DOWNAME.	Monday	EURDFDWN.	maandag
MONNAME.	May	EURDFMN.	mei
MONYY.	MAY04	EURDFMY.	mei04
WEEKDATX.	Monday, 10 May 2004	EURDFWKX.	maandag, 10 mei 2004
WORDDATX.	10 May 2004	EURDFWKX.	maandag, 10 mei 2004
WEEKDAY.	2	EURDFDN	1

THOUSANDS SEPARATORS, DECIMALS AND CURRENCY SIGNS

Within the US and most of Canada, numeric data is generally displayed with commas as thousands separators and a period denoting the decimal portion of the number, e.g. 123,456.78. However, right here in Quebec and much of Europe, the roles of the comma and period are reversed, e.g.123.456,78 SAS has provided a format for this notation. Rather than specifying **comma10.2**, use **commax10.2**.

```
a = 123456.78;
put a comma10.2 / a commax10.2;
put a dollar11.2 / a dollarx11.2;

123,456.78
123.456,78
$123,456.78
$123.456,78
```

While **dollarx.** flips the commas and periods in the same fashion as **commax.** it doesn't quite satisfy Quebec or some European countries. In these locales, the currency sign is placed following the number. In **v9** new currency functionality has been introduced to provide much more flexibility in currency formats. (*examples provided in presentation*)

SUMMARIZE DETAIL USING FORMATTED VALUES

Base SAS comes with many useful procedures designed to make common computing tasks more efficient; more efficient both in terms of coding and processing effort. As a rule, if SAS has provided a procedure to deal with a data manipulation task, it really ought to be used over a home-grown solution. For example, the TABULATE procedure displays descriptive statistics (e.g. mean, sum) in tabular format with very little coding effort. Consider a small data set containing sales figures by date.

(Note: date field literals defined with the '*date value*'**d** syntax are *automatically converted* to internal SAS date values. i.e. the do loop iterates from 15706 to 16070, the internal values for 01Jan2003 and 31Dec2003)

```
data analysis ;
  format date   yymmddd10.
         sales  dollar12.2;
  do date = '01jan2003'd to '31dec2003'd by 11;
      sales  = ranuni(1) * 20000;
      output;
  end;
run;
```

Since the **date** variable contains internal SAS date values, the benefit of various SAS date formats can be brought to bear on the field, even though a permanent format was defined in the data step. The simplest way to summarize the sales figures by month and quarter is using the simple TABULATEs below. Note that no pre-processing of the data was required and the only differences between the TABULATE steps is the **format** specified for the **date** variable.

```
proc tabulate data=analysis;
   class date;
     var sales ;
   table date='Month / Year',sales='Sales' *
sum=' ' *f=dollar12.2;
 format date monyy5.;
run;
```

Note that the **monyy5.** format has been applied to the date field, thus summarizing the sales data by month and year.

Month / Year	Sales
JAN03	$31,097.51
FEB03	$43,005.57
MAR03	$22,489.30
etc...	
OCT03	$41,397.52
NOV03	$33,759.13
DEC03	$47,338.74

```
proc tabulate data=analysis;
   class date;
     var sales ;
   table date='Quarter',sales='Sales' *
sum=' ' *f=dollar12.2;
  format date yyq6.;
run;
```

Note that the **yyq6.** format has been applied to the date field, thus summarizing the sales data by year and *quarter*. Same dataset, same field, same data, but different results via altered format specifications.

```
-----------------------------------
|                      |   Sales    |
|--------------------- +------------|
|Quarter               |            |
|--------------------- |            |
|2003Q1                |  $96,592.38|
|--------------------- +------------|
|2003Q2                |  $77,143.28|
|--------------------- +------------|
|2003Q3                |  $86,315.43|
|--------------------- +------------|
|2003Q4                | $122,495.39|
-----------------------------------
```

TABLE OF (SOME) INTERESTING (IN)FORMATS

Raw Data	**Informat**	**Result**
43414E414441	$HEX12.	CANADA
78.4%	PERCENT5.	.784
(25%)	PERCENT5.	-.25
2004131	JULIAN7.	16201
10May2004:13:30:00.0 0	DATETIME18.	1399815000
'3100320033'x	$CHARZB5.	1 2 3
"Quoted Value"	$QUOTE.	Quoted Value
2004Q3	YYQ6.	16253
SAS Data	**Format (note different lengths)**	**Result**
16201	YYMMDDS10.	2004/05/10
16201	MMDDYYD10.	05-10-2004
16201	MMDDYYD8	05-02-04
16201	DOWNAME.	Monday
16100	MONNAME10.	January
16100	MONNAME3.	Jan
16100	WORDDATE.	January 30, 2004
-10.4	NEGPAREN6.1	(10.4)
1300008	E7.	1.3E+06
1998	ROMAN10.	MCMXCVIII
-15	WORDS20.	minus fifteen
123	BINARY.	01111011
ACDC	$HEX8.	41434443
ONTARIO	$2.	ON
SAS	$REVERS.	SAS :-)

$CONCLUSION.

Sometimes it seems our working life consists of nothing but data, mountains of it. SAS is our favorite tool for scaling that mountain, reading and massaging data, slicing, dicing, summarizing, storing and finally presenting it again in a meaningful manner. SAS informats and formats, powerful instructions in and of themselves, are very helpful in all those activities.

Informats allow us to more easily convert the most mundane or the strangest raw data into SAS usable form. With each release of SAS, more formats are being added to present data in an ever-increasing myriad of output choices. But formats are not only for the start and end of our data journey, there's many applications along the way. There's power in those little guys with the mysterious dot!

Processing_Power = input(raw_data, informats.);
Presentation_Flexibility = put(SAS_data,$formats.);
Programmer_Efficiency = Processing_Power + Presentation_Flexibility

REFERENCES

SAS-L, many posts over many years. If you don't know what **SAS-L** is, run, don't walk, to your computer and enter http://listserv.uga.edu/archives/sas_-l.html in your browser address bar. Subscribe, browse, search the archives, learn.

SAS Online Documentation, Version 8, SAS Institute.

ACKNOWLEDGMENTS

Thanks to Section Chairs Debbie Buck and Helen-Jean Talbott for inviting me to present this paper. Appreciation to Marje Fecht of Prowerk Consulting (http://www.prowerk.com) for her ideas and proof-reading patience.

CONTACT INFORMATION

Comments and questions are encouraged. The author may be contacted at:

Harry Droogendyk
Stratia Consulting Inc.
PO Box 145
Lynden, ON L0R 1T0
519-647-2472
sugi@stratia.ca

LEGAL STUFF