Paper 101-27

Have a Strange DATE? Create your own INFORMAT to Deal with Her

Venky Chakravarthy, Ann Arbor, MI

ABSTRACT

Whatever the title may lead you to believe, this is a serious discussion of ***dates*** that are ***strange*** to the SAS® Software. You will read about date values that cannot be readily produced with the supplied date FORMATS and some that are not recognized by the supplied date INFORMATS. Enhancements to the date FORMATS enable you to customize dates in several ways. While these enhancements are welcomed, there are some limitations when it comes to inputting date values. Date INFORMATS, which are the tools for converting to SAS date values, are lagging behind in comparison. This paper deals with date values that are out of the range of current SAS date FORMATS/INFORMATS and suggests ways to output/input such values. The bulk of the discussion, however, focuses on how to INPUT such date values and how to create your own INFORMAT to do so.

INTRODUCTION

Despite the frivolous title of this paper and some of the headers within, this is a serious discussion. You, as a SAS user, often deal with date values. Some times you are reporting a date value in a form that can be supplied with an available SAS FORMAT and at other times you are reading in a date value with a SAS date INFORMAT. You are also (often) required to report a date value in a form that cannot be produced using a SAS supplied date FORMAT. Likewise, you are also trying to read in a date value for which there is no SAS date INFORMAT. In short, the form a date value can take is limited only by the imagination of your customer. You have to get creative in dealing with date values that a FORMAT/INFORMAT cannot readily handle. Consider one poster on the SAS-L forum who asked, "I was wondering whether anyone had a solution to read the following data line using a SASINFORMAT: December 20, 1999". While the WORDDATE format can OUTPUT such a value, there is no corresponding SAS date INFORMAT to INPUT such a value. How do you then input this date value?

This paper starts with a discussion of dates that are simple to deal with. Progressively, it discusses ways to write and read date values that are beyond the scope of ready handling with existing FORMATS and INFORMATS. The paper also discusses how to create INFORMATS to input such date values. Finally, it compares some of the merits and demerits of creating INFORMATS with another method of inputting such dates.

Only, a basic understanding of SAS date FORMATS and INFORMATS is assumed. Also assumed is an awareness of the SAS date FUNCTIONS and some familiarity with PROC FORMAT. The target audience is both the beginner and the advanced user.

THE GOOD DATES

Let us begin with a simple output of today's date, such as April 17, 2002 using the WORDDATE format mentioned above:

Code #1

```
1. data _null_ ;
2.    SUGI_day = "17apr2002"d ;
3.    put SUGI_day = ;
4.    put SUGI_day : WORDDATE. ;
5. run ;
```

Line #2, instructs SAS to treat the value in as a date value. Line #3 writes "SUGI_DAY=15447" to the SAS log. This means that this day of SUGI 27 is the 15447th day after January 1, 1960. SAS stores date and time values as deviations from this latter date. For example December 31, 1959 is stored as a numeric type with a value of negative 1 (-1), meaning 1 day before January 1, 1960. This type of storage facilitates calculations between date values such as days, months, years, etc.

In the same code above, line #4 instructs SAS to apply the WORDDATE format to the numeric value of 15447 and output the result to the LOG. This produces "April 17, 2002". There are several formats available. Without going into an exhaustive discussion of the various DATE FORMATS, Table 1 below captures ten other ways the same date can be output. The associated DATE FORMATS are also reported.

Table 1: Some FORMATS

Form of OUTPUT	Associated SAS FORMAT[1]
04/17/2002	MMDDYY10
17.04.2002	DDMMYY***P***10
2002:04:17	YYMMDD***C***10
Wednesday	DOWNAME9
April	MONNAME7
APR2002	MONYY7
Wednesday, April 17, 2002	WEEKDATE30
4	WEEKDAY1
17	DAY2
17APR2002	DATE9

You may be familiar with most of the above. If you are new to SAS versions that are post-V6.12, the separators (e.g. colon) that can be used in the FORMATS may come as news to you.

You have been refreshed with the FORMATS that are used to output dates in a number of ways. How would you like a refresher to some of the existing INFORMATS? Let us do this now:

HOW TO READ A GOOD DATE

Let us begin by reading today's date as though it is found in an external file:

Code #2

```
1. data _null_ ;
2.    extvalue = "4172002" ;
3.    SUGI_day = input(extvalue,mmddyy10.) ;
4.    put SUGI_day = ;
5. run ;
```

The variable EXTVALUE, in line #2 above, is the text string that you would read from an external file. In line #3 the INPUT function instructs SAS to apply the MMDDYY10 INFORMAT to read the string as a date value. Finally line #4 writes the value 15447 to the log. This demonstrates a successful conversion of a text string to a SAS DATE value. A brief list of INFORMATS that

[1] A separator can be specified for some of the formats to separate the month, day and year values. Valid separators are Blank, Colon, Dash, No separator, Period and Slash. The first letter of each is used as *italicized* and **blocked** in the table.

is associated with the corresponding INPUT string is presented in Table 2.

Table 2: Some INFORMATS

Form of INPUT	SAS INFORMAT[2]
17apr2002	DATE9
17 04 2002	DDMMYY10
41702	MMDDYY
2-4-17	YYMMDD
02-4-17	YYMMDD7
APR 2002	MONYY8
024	YYMM***N***
2002Q02	YYQ9
17apr2002/10:10 AM	DATETIME20

The above table gives an idea of the INFORMATS that can be used to read a value as a SAS date.

In general it appears that there are fewer INFORMATS than FORMATS to deal with DATES. The gap seems to have widened with the versions after 6.12. The FORMATS have received some nice enhancements. The YYMMDD***XW***, DDMMYY***XW***, and MMDDYY***XW*** FORMATS provide instructions to output date values with a choice of separators (see table 1 for examples). The DIRECTIVES in the PICTURE statement that were added to the arsenal of FORMATS give you some powerful tools. Perhaps, these enhancements were targeted towards report generation. Maybe, customers demand date values in increasingly different forms.

We have observed the other enhancements to the FORMATS in Table 1 but not the DIRECTIVES in the PICTURE statement. Let us examine this in the next section.

SOME DATES NEED DIRECTIVES

You have so far seen dates that could be easily dealt with. Suppose you had a request to output today's date as APR 2002. In version 6.12 this would have led to some messy coding. In version 8.2, although it entails creating a user defined FORMAT, the solution is relatively clean using the DIRECTIVES in the PICTURE statement:

Code #3

```
1. proc format ;
2.    picture myfmt low-high = '%b %Y'
3.          (datatype = date) ;
4.  run ;

5. data _null_ ;
6.    mydate = "17apr2002"d ;
7.    put mydate = : myfmt8. ;
8.  run ;

LINE #7 LOG:  MYDATE=APR 2002
```

Line #2 in Code #3 carries the DIRECTIVES, which are the instructions with the % sign. Note that these are case sensitive. For example the %Y (the capital Y) outputs the year with century as a decimal. The %y (the small y) on the other hand would output the year without the century as a decimal value with no leading zero. However, a leading zero can be output when you use %0y as a DIRECTIVE. The DATATYPE= option must be specified as in line #3 for the directives to work. In line #7 the format is called with a specific width (MYFMT8.) to be applied to the variable mydate. This width specification ensures that the assigned PICTURE format is wide enough to print the results to the log without truncation. Alternatively, a sufficient number of blanks can be embedded in the directive to achieve the same result (see line #3 in code #4 below). However, this gets tedious with greater widths and it is simpler to specify the width (maximum is 40). Code #4 below demonstrates some more uses of the DIRECTIVES used in the PICTURE statement.

Code #4

```
1. proc format ;
2.    picture brief low-high =
3.       '%dth %b is a %a    '
4.       (datatype = date) ;
5.
6.    picture long low-high =
7.       '%dth %B is a %A'
8.       (datatype = date) ;
9. run ;
10.
11. data _null_ ;
12.    sugiday = "17apr2002"d ;
13.    put sugiday : brief. /
14.        sugiday : long40. ;
15. run ;

LINE #13 LOG: 17th APR is a Wed
LINE #14 LOG: 17th April is a Wednesday
```

The day of the week can be written to a report for any date in the calendar. We can span a couple of centuries without much fuss. The two picture formats demonstrate the case sensitivity of the DIRECTIVES as seen in the LOG.

A LIST OF DIRECTIVES FOR YOUR DATES

Figure 1 below contains a short list of useful PICTURE FORMAT DIRECTIVES (Lund, 2001), that aid in outputting date values. Please note that in combination with the possibility of including text as in code#4 these DIRECTIVES can produce almost any kind of date value you need.

Figure 1

```
1.  %A Full weekday name
2.  %b Abbreviated month name
3.  %B Locale's full month name
4.  %d Day of the month as a decimal number
5.  %j Day of the year as a decimal number
6.  %m Month as a decimal
7.  %w Weekday as a decimal number
8.  %y Year without century
9.  %Y Year with century
```

Now that we have seen the good dates and the not so good dates it is time to turn to the strange dates - the main focus of this paper.

THE STRANGE DATES

Some dates are absolutely strange. There is no simple way to deal with them. This is despite the enhanced capability of Version 8.2 demonstrated above. Especially so, with reading some date values provided by an external vendor. Other software packages sometimes write out date values in forms that are not recognized by a readily available INFORMAT. The forms date values can

[2]The YYMM***N*** INFORMAT comes with a specific instruction (the ***N***) to show that there can be no delimiters between the Year and the Month values. It also adds 01 as the day to make it a valid SAS date value.

take appear to be limitless. This section will deal with such date values that do not have a readily available INFORMAT to deal with them. We will call this our strange dates.

Let us go back to the first example in Code #1 and look at the output. The WORDDATE FORMAT was used to output today's date as "April 17, 2002". Let us assume that a vendor supplied ASCII text file has a date value in such a form. Now, how do we input such a date value (also the SAS-L question mentioned in the INTRODUCTION)? We have noted before that to read a date value we must apply a DATE INFORMAT. Unfortunately there is no corresponding WORDDATE INFORMAT to readily input this date value. Let us examine how we can work around this inadequacy.

The solution that readily comes to mind is to read the value as a character field, strip it apart, and put the individual parts together while applying some FUNCTIONS, FORMATS and INFORMATS. Let us do this in code #5:

Code #5

```
1. data _null_ ;
2.    _txtdate = "April 17, 2002" ;
3.    _txtdate = trim ( left (_txtdate) ) ;
4.    _s1 = scan( _txtdate,2,', ') ;
6.    _s2 = substr(
7.              scan(_txtdate,1,', '),1,3) ;
8.    _s3 = scan(_txtdate,3,', ') ;
9.    _sasdate = input(put(_s1,2.)||
10.                   put(_s2,3.)||
11.                   put(_s3,4.),date9.);
12.  put _sasdate ;
13. run ;

LINE #12 LOG: 15447
```

Although ugly, this is an acceptable working solution. Line #3 flushes the text string in _TXTDATE to the left and trims any trailing blanks. Line #4 scans and reads the value between the first and the second delimiter. This is "17", the day, and is stored as a character value in _s1. Line #5, likewise scans the value until the first delimiter and extracts the first three letters. This is the month value "Apr" and is stored as a character value in _s2. Similarly the year value of "2002" is extracted and stored in _s3. Finally, these three pieces _s1, _s2 and _s3 are concatenated and the DATE9. INFORMAT is applied as an input to convert the character values to numbers. Now that the principle has been explained, code #6 below simplifies this further:

Code #6

```
1. data _null_ ;
2.    _txtdate = 'April 17, 2002' ;
3.    _txtdate = trim ( left (_txtdate) ) ;
4.    _sasdate=input (
5.               scan(_txtdate,2,', ') ||
6.               substr(scan
7.                (_txtdate,1,', '),1,3) ||
8.               scan(_txtdate,3,',') ,
9.               date9.) ;
10.   put _sasdate ;
11. run;

LINE #10 LOG: 15447
```

We have observed in code #5 how the literal date was stripped apart and then the pieces put together with the INPUT function using the DATE9 INFORMAT. Code #6 eliminated the variables: _s1, _s2 and _s3 by nesting what they did within the INPUT function. The code#6 also eliminated some of the overheads such as multiple PUT functions.

Although code #6 is less readable than code #5, there are some nice things that can be said about this code. It provides more parsimony by eliminating the creation of a few extra variables and functions. It executes slightly faster than code #5 (the relative performances of the different codes are discussed elsewhere in this paper). One line (albeit long) of code does the job of four different lines of code. This code is good for reading in small files and especially if it is a one-time situation.

There are also some things that make this approach less desirable.

(1) In real world applications it is common to find situations where a few million such dates have to be read. There are overheads created by the INPUT, TRIM, LEFT, SCAN and SUBSTR functions. These overheads increase the time of processing and may be scorned upon in a production environment.

(2) It is also common to find junior programmers dealing with such data. A solution such as the above may be fairly simple to arrive at for an advanced user but may be a daunting task for someone who is relatively inexperienced. If one searches the SAS-L archives one can find a number of postings asking how to deal with such unusual dates.

So, how can we find a solution that can be readily applied by junior programmers and that also executes within a reasonable time frame? The next section addresses this.

CREATING YOUR OWN INFORMAT

The solution that addresses the concerns raised above may lie in a user created INFORMAT[3]. If the INFORMAT isn't available to you then you must create the INFORMAT. Once a USER defined INFORMAT is created and stored it can be called like any other supplied INFORMAT and applied to a value. The following solution (code #7 through #9) demonstrates how to create a USER defined INFORMAT and apply it to such date values:

Code #7

```
1.  data infmt ;
2.    retain fmtname "monddyr" type  "I" ;
3.    do label = "1jan1999"d to
4.                "1jan2003"d ;
5.       start = put (label,worddate.) ;
6.       start = trim ( left (start) ) ;
7.       output ;
8.    end ;
9.    run ;
```

We have created a data set with variables FMTNAME, TYPE, LABEL, and START (code #7). We can readily recognize this as a typical control SAS data set used in PROC FORMAT to output user defined FORMATS and INFORMATS. Readers not familiar with control data sets can review the PROC FORMAT documentation. The variable LABEL in line #3 is created as a SAS date value iterating through the range specified. The values of LABEL and START are written to the output data set, for each iteration, of the DO LOOP. Notice that the available WORDDATE format has been used to simplify the creation of the values for the variable START. The value of the variable TYPE is "I". This indicates that an INFORMAT will be output in code #8 below.

[3] User defined INFORMAT names can have a maximum length of seven (7) characters only. This is unlike any other user-defined name where the maximum length is 8 or above.

Code #8

```
1.  proc format cntlin = infmt ;
2.  run ;

LOG NOTE: Informat MONDDYR has been output.
```

The LOG confirms that a USER defined INFORMAT named MONDDYR has been output. Now it is ready to be used. Code #9 demonstrates how this INFORMAT can be used.

Code #9

```
1.   data _null_ ;
2.     _txtdate = 'April 17, 2002' ;
3.     _sasdate = input(_txtdate,monddyr.) ;
4.     put _sasdate ;
5.   run ;

LINE #4 LOG: 15447
```

Although, at first, this coding scheme appears tedious, it has some advantages over the previous solutions.

1. It takes very little time to create a control data set for the PROC FORMAT.
2. The range of date values can span a few decades or even a few centuries if needed.
3. The control data set in its end form acts as a LOOK-UP table.
4. All the undesirable overheads of using multiple SAS functions are eliminated at the time of conversion.
5. The code that does the actual job of reading such a date value is very simple.
6. A junior programmer can be made aware of the name of the INFORMAT stored in a CATALOG.
7. This results in the one line solution in Line #3 of Code #9 to solve his strange date problem.
8. Creating and using an INFORMAT also makes the code a great deal more robust and portable.
9. Other INFORMATS can be created and stored in a common library.

You may recall that the PICTURE FORMAT DIRECTIVES were created in a similar way as a user defined format and then applied later to output values. The basic principle is the same.

The obvious disadvantage of this code is that it takes some amount of experience with PROC FORMAT and a good understanding of ways to improvise. For example, a beginner's effort to create the variable START in line #5 of code #7 may be a bit involved. Suppose this paper did not include code #1 to demonstrate that a WORDDATE FORMAT was available. It would be a rare beginner to intuitively think of applying an available DATE FORMAT while creating a control data set for a DATE INFORMAT?

You have just looked at a date value from an external file and created your own INFORMAT to read it as a SAS date value. Fortunately for you the creation of the control data set was simplified by the availability of an equivalent FORMAT. This was intentionally done as an illustration to make a simple beginning. However, in many real life applications such conveniences may be happenstance. It is more realistic to expect difficult cases that may involve some more coding.

Suppose you have a file with date values in the form "2002/APR/17". Using the above principles we could output a control data set. You will recognize that a ready FORMAT cannot be applied to produce the control data set as the WORDDATE in code #7. The PICTURE FORMAT DIRECTIVES can be used to output a format and then use that format to write out a control data set.

Code #10

```
1.  proc format ;
2.     picture temp low-high =
3.     '%Y/%b/%d'
4.     (datatype = date) ;
5.  run ;

6.  data infmt ;
7.     retain fmtname "yyyymd" type "I" ;
8.     do label = "01jan1999"d to
9.                "01jan2003"d ;
10.       start = put(label,temp11.) ;
11.       start = trim (left (start) ) ;
12.       output ;
13.    end ;
14. run ;
```

A format named TEMP is output using the DIRECTIVES in the PICTURE statement. The only purpose of this format is to provide the ability to use this format in the next step.

The DIRECTIVE '%Y/%b/%d' in line #3 of code #10 is a powerful replacement for how you would have coded otherwise. Consider the following equivalent in earlier versions of SAS®:

Code #11

```
start = put(label,year4.) || "/" ||
        upcase(put(label,monname3.)) || "/" ||
        left(put(label,day2.)) ;
```

Once the control data set is created it is simply a matter of creating an INFORMAT out of it and call it when needed. This is done in code #12 below:

Code #12

```
1.  proc format cntlin = infmt ;
2.  run ;

3.  data _null_ ;
4.     _txtdate = "2002/APR/17" ;
5.     _sasdate = input (_txtdate,yyyymd.) ;
6.     put _sasdate = ;
7.  run ;

LINE #6 LOG: 15447
```

The issue of performance was mentioned before but a proof was referred to a later section. You will not be entering into a treatise on performance measures. This will follow a common sense approach to compare the performance of two competing approaches to solving your problem of dealing with dates. Let us look at this now.

STRANGE DATES: APPROACHES COMPARED

Your problem is simply one of converting a text string to a SAS DATE value. One approach stripped the text apart and put it back together applying a number of FORMATS and FUNCTIONS (approach A). The competing approach is to CREATE a user defined INFORMAT and apply it to the incoming text string (approach B). Before we compare the performance in terms of clock times, let us outline some key differences in the two approaches.

1. Approach A converts the value in a single step. Approach B

splits the conversion process into several steps.
2. Approach A has fewer lines of coding while Approach B has many lines of coding.
3. Approach A incurs the overheads of several functions and formats in one single step. Approach B on the other hand distributes the load into several steps.
4. The line that does the actual conversion is long and tedious in Approach A. The actual conversion in Approach B is a simple one-liner that uses the INFORMAT.

The differences outlined in #3 and #4 address our main concerns about a solution dealing with strange dates. You may recall that we wanted a solution that would execute in a reasonable amount of time and we also wanted a solution that is easy for a junior programmer to handle.

The difference outlined in #3 above appears the most logical determinant of performance. If the distribution of the load is judicious in approach B, then superior performance could be expected in real life applications involving millions of records.

The relative performances were evaluated based on ten million iterations that would simulate reading in 10 million date values in real life applications. This process was repeated multiple times to rule out any capitalization on chance that a freak run might result in (the author can be contacted for the code that compares the two approaches).

The results clearly indicate that the INFORMAT approach is superior in that it executed faster. On average, the INFORMAT approach executed in 38.42 seconds to process 10 million date values. The corresponding time for the approach involving stripping the values apart and putting them back together applying formats and functions was 83.17 seconds. In other words the INFORMAT approach executed in less than half the time taken by the other approach. In every single run of the experiment the INFORMAT approach executed at least twice as fast as the other approach. The results from a random selection of 5 repeats of the experiment are reported in Table 3 below. All numbers have been rounded to 2 decimal places.

Table 3: Performance Metrics (in Seconds except RATIO)

INFORMAT APPROACH	OTHER APPROACH	DIFFERENCE INF - OTH	RATIO INF/OTH
38.16	82.13	-43.97	0.46
39.92	84.42	-44.50	0.47
37.91	81.75	-43.84	0.46
38.14	81.95	-43.81	0.47
37.95	81.69	-43.73	0.46

The other difference outlined in #4 addresses our other concern. We wanted a solution that would be suitable for a junior programmer. In the user defined INFORMAT the first few steps are done only once. These are the difficult ones and can be done by an experienced programmer. This creates the INFORMAT, which can be stored in a permanent library (left to you as an exercise). Thus a junior programmer has to write only lines #3 through #7 of code #12. This effectively addresses our second concern.

LIMITATIONS OF THIS PAPER

This paper discussed two common approaches to handling date values that are beyond the scope of supplied FORMATS or INFORMATS. There may be other approaches. The solutions are also not intended to cover all possible date types. Time values have been left out of the discussion. If instead of date values, you have date and time values, then creating an INFORMAT may be impracticable (why? HINT: number of seconds in a day). The INFORMAT solution should not be considered as the golden rule because it is not always good for all situations. Although, performance wise, the user defined INFORMAT approach was demonstrated to be superior there are other factors involved that may make the other approach more practicable. If you only deal with a couple of hundred and not a million observations or you do not deal with this type of situation often then the first solution that comes to mind is the most efficient for you.

SUMMARY

We have come to the end of this discussion so let us summarize the main points.

1. Many date FORMATS and INFORMATS are supplied with SAS that can deal with a good range of date values.
2. The FORMATS have received some nice enhancements, mainly through the PICTURE FORMAT DIRECTIVES.
3. The date INFORMATS have lagged behind.
4. Reading date values outside the scope of the supplied INFORMATS is possible but requires special handling.
5. One approach reads the value as a text string, splits the string into pieces and applies FORMATS and FUNCTIONS while concatenating the pieces together and then inputting the whole with an existing INFORMAT (see code #5 and #6).
6. A second approach creates a user defined INFORMAT to apply to the input string (see code #7 through #12).
7. The user defined INFORMAT approach outperformed the other in two main ways: it executed faster and made it simple for a junior programmer to input the value.

CONCLUSION

Creating your own INFORMATS do deal with bad date values seem to be the preferred solution. The exceptions have been outlined in the Limitations section. The discussion further emphasizes the point that the INFORMATS have not been given the same enhancements that the FORMATS have received. While we can speculate on the reasoning, it is more important to get this on the ballot for enhancements in future versions. I would recommend asking for the equivalent of the PICTURE FORMAT DIRECTIVES.

REFERENCES

Lund, P. (2001), “More than Just Value: A Look into the Depths of PROC FORMAT,” *Proceedings of the Twenty sixth Annual SAS Users Group International Conference*, 18, 1-10.

ACKNOWLEDGMENTS

I thank Ian Whitlock whose suggestions were invaluable. I also thank Paul Dorfman for his input on measuring the performance of competing codes. Also, I thank the many SAS-L posters whose questions prompted this paper.

CONTACT INFORMATION

Your comments and questions may be sent to:

Venky Chakravarthy
1591 Abigail Way
Ann Arbor, MI 48103
(734) 622 - 1963
swovcc@hotmail.com