A Technique for Storing and Manipulating Incomplete Dates in a Single SAS Date Value

John Ingersoll

Introduction:

This paper presents a technique for storing incomplete date values in a single variable and methodologies for processing them. A limitation of SAS date values is the assumption that the month, day, and year of the date are all known. In many real world situations date data are often incomplete, with only a month and year known, or just the year. Often, however, a partial date is adequate for the purpose or application, but one is left to program around the missing date parts, losing the built in benefits of the SAS date processing facilities. The solution presented here allows you to store date values, whether complete or partial, in a single variable that you can manipulate using all of SAS's date processing tools, while allowing you to identify and report dates with missing parts. The conceptual framework of this approach addresses the issues of lost or inaccurate information, data redundancies, additional storage requirements, and processing inefficiencies associated with the methods commonly used to deal with incomplete dates. Basic data step processing techniques for applying this solution are presented, in addition to macro tools specifically designed to handle partial date values in a manner consistent with typical SAS programming.

Background:

SAS stores date values as numeric variables, as integers representing the number of days from 1/1/1960. A SAS date variable is not a distinct variable type, as in other database management systems, but simply a numeric variable that is distinguished by its use. Often a date format will be permanently associated with the variable to assure that it prints as a date and not an integer, and also to identify it as a date value.

One advantage of this approach that SAS uses is that simple arithmetic may be used to manipulate date variables. The number of days between two dates is simply the difference in their respective values. The number of months and years can closely estimated by dividing by the average number of days in a month or year, respectively.

Example 1:

```
data one;

   date1 = '23Feb92'd;
   date2 = '04Mar93'd;

   days = date2 - date1;
   months = days / 30.4375;
   years = days / 365.25;

   put days=;
   put months=;
```

```
   put years=;

run;
```

Result:

```
days=375
months=12.320328542
years=1.0266940452
```

SAS also provides a plethora of formats, functions and tools for manipulating date values which can be invaluable to the programmer while processing dates.

The Problem:

All of SAS's wonderful date processing features, however, are lost as soon as your data contains incomplete dates, because a SAS date value requires that the month, day, and year must all be known. There are several ways these incomplete dates are typically dealt with:

	Solution	Disadvantages
a.	Abandon any incomplete dates and set the SAS date value to missing.	You lose information. In many cases the month and year (or only the year) is sufficient for the analysis at hand.
b.	Store the date in 3 separate month, day, and year variables.	Cumbersome both in terms of data storage and manipulation. Maintain 3 variables instead of 1. Lose the use of all SAS date processing features.
c.	Store dates as character variables.	You cannot perform any calculations or manipulations on the values.
d.	Substitute values for missing date parts and create a SAS date value based on the best information available.	You lose information and the ability to distinguish real from incomplete dates.

Commonly a mix or combination of these approaches is used, but they all become cumbersome. If all methods are used, you could end up with 9 or more variables to store information on a single date value. When used together, these methods both retain all available information and maximum flexibility in potential processing options and decisions on use, but it greatly increases storage requirements, adds complexity to the database, and can be difficult to program around.

Variable	Type	Rule
Raw_Date	SAS Date Value	Missing value if any missing date part.
Raw_Date_Char	Character	Character representation of date.
Raw_Month	Integer	Missing value if missing month.
Raw_Day	Integer	Missing value if missing day.

Raw_Year	Integer	Missing value if missing year.
Month_with_Substitution	Integer	1 if Raw_Month is missing.
Day_with_Substitution	Integer	1 if Raw_Day is missing.
Year_with_Substitution	Integer	2002 if Raw_Year is missing.
Date_with_Substitution	SAS Date Value	Using Substitution variables.

The Partial Date Method:

The solution presented here allows you to store date values, whether complete or partial, in a single variable that you can manipulate using all of SAS's date processing tools while allowing you to identify and report dates with missing parts.

The partial date method employs a substitution approach without the overhead of multiple variables to store partial information. The "trick" it uses relies on the fact that although SAS date values are integers, they are stored as numeric variables and a decimal portion can be used as a flag to identify missing and substituted parts. SAS simply ignores the decimal part of the number when you apply the date formats and functions to a non-integer number.

Take the example of 1/15/60. The integer value that SAS assigns to this date is 14. Suppose, however, that the source data was simply January 1960 and did not include the day of the month. We could substitute the 15th of the month for the missing day part and store the date as a SAS date value (i.e. 14). But if that's all we do, we lose information because won't know that 1/15/60 is an estimate based on incomplete information. What we can do, however, is to add 0.1 to the date value and store the value as 14.1. SAS basically ignores the decimal place when doing any date processing, but we now have the ability to test and identify the data value as a partial date that only truly represents the month and year.

Source Data	**Stored Numeric Value**	**SAS Formatted Result**	**PDM Formatted Result**
15JAN1960	14	01/15/1960	01/15/1960
xxJAN1960	14.1	01/15/1960	01/../1960

To operationalize this methodology you must be able to:

(a) Identify missing date part(s);
(b) Substitute valid values for missing parts;
(c) Create a valid SAS date value;
(d) Flag the result so you can identify what's real and what's been substituted for; and
(e) Create tools to test and report partial date values.

Reading Incomplete Dates and Storing the Values:

There are many possible approaches to identifying incomplete data, substituting for the missing parts, and storing the decimal flags with the date values. In every case there are certain

assumptions about the data that have to be made and decisions made about those assumptions. The example below, for instance, assumes that the raw data are in strict column format with each portion of the data in a specific column. It also assumes that the dates are delimited and have 4 digit years.

The decimal place that is assigned represents the following:

.1 Day is missing
.2 Month is missing
.3 Day and Month are missing

The substitution rule for day of the month is to use the 15th. The substitution rule for a missing month is to use July 1st, whether or not the day is missing. The decision was made that a day without a month is not meaningful, so the middle of the year is used in either case. Similarly, if the year is missing then we set the date value to missing. Any number of different substitution rules could be defined for different applications, as is appropriate. This is only one example.

Example 2:

```
data two;

   *--------------------------------------;
   * Read the Raw Data as a character value ;
   *--------------------------------------;
   input @1 char_date $char10. @12 description $30.;

   *-------------------------------------------------;
   * Test character date representation for valid date ;
   *-------------------------------------------------;
   PDM_date = input( char_date, ??mmddyy10. );
   if PDM_date > . then goto GotDate;

   *-------------------------------------------------------------;
   * Test for a valid Year value                                 ;
   * - Assume that the year must be valid and do not substitute ;
   *-------------------------------------------------------------;
   char_year = substr( char_date, 7, 4 );
   num_year  = input( char_year, ??4. );
   if num_year = . then goto GotDate;
   if 99 < num_year < 1582 then goto GotDate;

   *-------------------------------------------------------------;
   * Test for a valid Month value                                ;
   * - Substitute July 1 if month is missing or invalid         ;
   *-------------------------------------------------------------;
   char_month = substr( char_date, 1, 2 );
   num_month  = input( char_month, ??2. );
   if not( 1 <= num_month <= 12 ) then do;
      PDM_date = mdy( 7, 1, num_year ) + 0.2 ;
      goto GotDate;
   end;

   *-------------------------------------------------------------;
   * Test for a valid Day value                                  ;
   * - Substitute the 15th if day is missing or invalid         ;
   *-------------------------------------------------------------;
```

```
   PDM_date = mdy( num_month, 15, num_year ) + 0.1 ;

   GotDate:

   PDM_date_fmt = put( PDM_date, mmddyy10. );

cards;
12/31/2003 Complete Date
04/04/1904 Complete Date
01/15/1960 Complete Date
07/01/1960 Complete Date
01/  /1960 Missing Day
xx/15/1960 Invalid Month
      1960 Missing Day and Month
01/15/1234 Invalid Year
09/31/2002 Invalid Day of Month
run;

proc print;
   var description char_date PDM_date PDM_date_fmt;
run;
```

Result:

Obs	description	char_date	PDM_date	PDM_date_fmt
1	Complete Date	12/31/2003	16070.0	12/31/2003
2	Complete Date	04/04/1904	-20360.0	04/04/1904
3	Complete Date	01/15/1960	14.0	01/15/1960
4	Complete Date	07/01/1960	182.0	07/01/1960
5	Missing Day	01/ /1960	14.1	01/15/1960
6	Invalid Month	xx/15/1960	182.2	07/01/1960
7	Missing Day and Month	1960	182.2	07/01/1960
8	Invalid Year	01/15/1234	.	.
9	Invalid Day of Month	09/31/2002	15598.1	09/15/2002

Performing Operations with Partial Date Values:

Once you have created the partial date values, you may use them as you would any SAS date value—SAS simply ignores the decimal portion that you have added. For those who are thinking, "What happens with dates prior the 1/1/1960, which are negative, of course", SAS in essence applies a FLOOR() function the values before processing. If you're so inclined, run the above example using earlier dates and see what happens.

But this does bring up the issue of using arithmetic on date values. When use the Partial Date Method, you must remember that you are dealing with decimal places and not just integers. Therefore, you should always apply the FLOOR() function before performing an arithmetic operation. The INT() function will give you erroneous results for dates prior to 1/1/1960. You could use the ROUND() function with examples presented here, because all flags used are <.5, but if the decimal portion is >.5 you *must* use the FLOOR() function.

Suppose you have 2 date values: DATE1 is a partial date of Jan.1960 to which we substituted the day of the month. DATE2 is the complete date of Jan. 16, 1960. The numeric values stored for them are 14.1 and 15, respectively. Consider the following results:

```
difference1 = date2 – date1;
difference2 = floor(date2) – floor(date1);
difference3 = intchk(date2,date1,'DAY');
```

DIFFERENCE1 will give an incorrect result of .9, which SAS will then treat as 0 if you apply any date function or format to it. DIFFERENCE2 and DIFFERENCE3 will both give a correct result of 1. Perhaps the best advice is to do away with arithmetic date programming all together and use the functions that SAS provides.

Reporting Partial Date Values:

In some situations reporting partial dates as complete dates with substitutions is appropriate and in other cases it is not. Therefore, we need to be able to test and conditionally format the dates. Since there is not a format that will automatically do this outside of the data step, it is necessary to create a character variable to hold the formatted date value prior to reporting. In the example below two dots ("..") are substituted for missing days and/or months.

Example 3:

```
data three;
   set two;

   report_date = put(PDM_date,mmddyy10.);
   if round(PDM_date-floor(PDM_date),.1) >= 0.1
      then substr(report_date,4,2) = '..';

   if round(PDM_date-floor(PDM_date),.1) >= 0.2
      then substr(report_date,1,2) = '..';

run;

proc print;
   format PDM_date mmddyy10.;
   var Char_date PDM_date report_date;
run;
```

Results:

Obs	char_date	PDM_date	report_ date
1	12/31/2003	12/31/2003	12/31/2003
2	04/04/1904	04/04/1904	04/04/1904
3	01/15/1960	01/15/1960	01/15/1960
4	07/01/1960	07/01/1960	07/01/1960
5	01/ /1960	01/15/1960	01/../1960
6	xx/15/1960	07/01/1960	../../1960
7	1960	07/01/1960	../../1960

```
8      01/15/1234              .              .
9      09/31/2002    09/15/2002     09/../2002
```

Macro Solutions:

There are many possible approaches to implementing the Partial Date Method, with a number of possible assumptions and decision rules that may be made. To implement it on a scale of any significance requires the development and implementation of macros to simplify repetitive programming and to maintain consistency in the application of the rules.

This overview of a set a macros that implement this methodology is intended to give the reader a sense of what is required and the type of considerations that need to be made. They will be provided as handouts and covered to the extent that time allows.

One key feature is that the macros are designed to mimic SAS functions as much as possible. That is, most return a SAS expression and are called on the left side of an equal sign. This design parameter was given priority and affects certain other assumptions and rules, particularly the last one in the list.

Assumptions and Rules:

- The macros are designed to be invoked similar to SAS functions and return parallel results.
- Raw data may be read as either 3 separate variables, or a single character string in a date representation. Character strings, however, must be in a positional format.
- If the Month is missing, then the Day is also substituted. Having a Day without a Month is considered extraneous information.
- Missing year parts are not dealt with in any special manner--the date value will be missing. Having a day or month without a year is considered meaningless information. In certain applications this may not be true, and an appropriate substitution could be defined.
- The 15th of the month is substituted for missing days.
- July 1st is substituted for missing months, regardless of whether the day is missing or not.
- A period (.) is used to represent missing date parts. Any character could be defined.
- Any day of the month that is 31 or less, but is not valid for that month and year (e.g. Sept .30th) will return a missing value for the date. A day of the month greater than 31 will return a partial date value.

The Tools:

%PDM_Initialize – A PROC FORMAT that defines informats and formats required by the other macros. This macro must be called prior to using any other PDM macros in a SAS job. Alternatively, the formats could be defined in a permanent format library provided that library is available to the SAS jobs.

%PDM_mdy – Imitates the MDY() function. Reads 3 separate variables into a SAS date variable, with substitution and flags.

%PDM_Input – Imitates the INPUT() function, for date informats only. Reads a single character variable or string into a SAS date variable, with substitution and flags.

%PDM_Put – Imitates the PUT() function, for date formats only. Creates a formatted date value with periods for missing date parts.

%PDM_PutSpec – Creates a specification that is valid in a PUT statement and writes a formatted date value with periods for missing date parts.

%PDM_IsDayMiss – Test whether a SAS date value has a missing day.

%PDM_IsMonthMiss – Test whether a SAS date value has a missing month.

%PDM_MissPart – Returns the decimal flag.

Conclusion:

The Partial Date Method provides the means to retain necessary information about a date value while simultaneously allowing the use of incomplete dates as regular SAS date values. It also reduces the overhead of maintaining multiple variables as is typically done when such information retention is required. However, to implement the Partial Date Method and the requisite tools on any significant scale is a business decision that will require buy in from all affected parties. All users of the data must be aware that the dates may not in fact be complete when they otherwise appear to be and they must also be knowledge of the techniques and tools for processing the incomplete data.

Contact Information:

Email: john.ingersoll@mindspring.com

Trademark Notice: