Date Manipulation Techniques, or How D'ya Show a Date a Good Time

Jerry Kagan, SmithKline Beecham Pharmaceuticals

Abstract

This paper is about date manipulation techniques. The main thrust will be a systematic approach to handling partial dates within the existing framework of SAS® date values. A history of the calendar system is given as an introduction to provide background. The concept of partial dates as usable information is examined. Sample macros for reading and printing partial dates are provided in the appendices. The ability to use partial dates is of particular interest in the pharmaceutical industry where incomplete dates are often encountered, but the concepts presented here would apply to any industry with a need to work with incomplete date information.

Introduction

One of the most frustrating aspects of programming for clinical trials is being forced to work with real world data. In some cases the data can be months, if not years, old. When data is incomplete or incorrect a query is issued to find the missing information. Not all queries are answerable, for a variety of reasons, and the programmer is often forced to make do with what is available.

Dates are among the most difficult items to work with when complete information is not available. Incomplete dates occur most often when participants of clinical trials are asked when they began prior medications, or when they first experienced symptoms of their primary diagnosis. Often, an exact date is not known and a partial date is recorded. One example would be, Jan 82 or just 1/82 as a partial start date. These partial dates make analysis difficult because they can not be directly converted to SAS date values for easy comparisons. Before a method to use partial date information is developed both the concept of dates and the SAS® system representation of dates will be reviewed.

How SAS Represents Dates

SAS date values have a valid range from 1 January 1582 through 31 December 20,000, almost 18,419 years. The range starts in the first year of the Gregorian calendar. Internally, SAS represents each day in the range as an integer on a number line, with 1 January 1960 as day 0. See figure 1. Leap years are handled automatically.

SAS datetime values are handled in a similar way, except that the integers are not the days from 1 January 1960, but rather the seconds from midnight, 1 January 1960. The range of valid datetime values is the same as date values. Leap seconds and daylight savings time, however, are not considered.

Possible Problems with Date Representations

The SAS system's method of representing dates is not standard. Other products use different methods. Oracle, for example, represents dates in a valid range from 1 January 4712 through 31 December 4712. Note that this extends out of the range of the Gregorian calendar.

Problems can arise when trying to exchange data between two systems that represent dates differently. Many companies use Oracle as a database and SAS as a reporting tool. With SAS/ACCESS®, data transfer is virtually transparent, but if you have values in Oracle that are not in the SAS date range, they will be passed to SAS as missing values.

A problem in my company arose when an Oracle database containing datetime values was accessed with SAS/ACCESS. We were looking only at the time portion of a datetime value for a lab test and could not understand why the values looked correct in Oracle, but when passed to SAS, a few of the values were missing. It turned out that the date portion of the value, which should have been set to the date of the lab test, was defaulting to the earliest Oracle date when the value was missing. While the time portion was present and was the only information we were interested in, the whole datetime value was reported missing because the date was out of range in SAS. This problem was almost overlooked because only a few values were reported missing. This problem illustrates the need to have an understanding of how the products you are using represent dates.

The Calendar System

The calendar system provides a basis on which we can compare one date with another. While we have all
grown up with a basic understanding of our calendar, the history and origin are not well known. Even the formal definitions of the common units, days, months, and years, are often poorly understood.

**Origin of the Gregorian Calendar**

Our calendar is simply a collection of rules on which a convenient chronology can be based. *Calendar* comes from the Latin word *Kalendae* which was the Roman word for the first day of the month. The calendar in general use today, known as the Gregorian calendar, was not simply invented, but rather evolved from older calendar systems, and has quite an interesting history. Although the Gregorian calendar is in almost universal use today, other calendar systems are still used on occasion.

**Calendar Units**

Most calendar systems use three basic units derived from the movements of the earth, moon and sun; in our system they are called the day, month, and year, respectively.

![Figure 1](https://example.com/figure1.png)

**Gregorian Dates**

<table>
<thead>
<tr>
<th>01 Jan 1582</th>
<th>01 Jan 1959</th>
<th>01 Jan 1960</th>
<th>01 Jan 1961</th>
<th>31 Dec 20,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>-138,081</td>
<td>-365</td>
<td>0</td>
<td>366</td>
<td>6,589,335</td>
</tr>
</tbody>
</table>

**SAS Date Values**

A day is often thought to be the measurement of one complete rotation of the earth about its axis. This is called a sidereal day, from the Latin, *sidus*, meaning star. Our calendar, however, is based on solar days, which are a measurement of the sun’s movement across the sky. Also called the civil day, the solar day is longer than the sidereal day by about 4 minutes because the sun’s position in the sky moves about 1 degree per day in the opposite direction to the motion of the stars.

Months were originally based on the length of a lunar month, which is the time the moon takes to complete a cycle of phases. The average length of a lunar month is 29.5306 solar days, which has a variance of ±6 hours.

Years are defined by the passing of the center of the sun across the equator from the Southern to the Northern hemisphere, called the vernal equinox. The time interval between two vernal equinoxes is called a solar, seasonal, or tropical year. The average length of a solar year is 365.2422 solar days.

Weeks are artificial units of time, not based on any natural phenomena. The days of the week were named after the most prominent heavenly bodies, the Sun, the Moon, Mars, Mercury, Jupiter, Venus, and Saturn.

**Calendar Types**

Only three basic types of calendars have existed; lunar, lunisolar, and solar.

Lunar calendars are based solely on the phases of the moon and always lag behind the solar year. The Islamic calendar is a good example of this type. It has 12 months that alternate between 29 and 30 days. In lunar calendar systems a fixed date will rotate through all of the seasons because of the differing lengths between the calendar year and the solar year.

Lunisolar calendars were developed once a good approximation of the length of the solar year was discovered. This calendar poses a problem because the lunar year is about 11 days shorter than the solar year, and keeping the calendar in alignment with both the season and the phase of the moon is difficult. Different cultures found different solutions to this problem. One solution was to add a 13th month to the calendar every 3 years although other methods were also developed.

Our current calendar system, the Gregorian calendar, is strictly solar. It still has 12 months, but the months are no longer tied to the phases of the moon. Its origin can be traced back to the Roman calendar, which, in turn, can be traced back to the Egyptian’s and the beginning of the Egyptian civilization, more than 10,000 years ago.
The names of the months of the Gregorian calendar come from the Roman calendar, which at first had been a lunisolar calendar based on a ten month year. The first month was March, the 7th, 8th, 9th, and 10th months were named September, October, November, and December, which are Latin words which indicate the position of the month in the year. According to legend, not until the reign of Numa Pompilius, about 700 B.C., were the months January and February added as the 11th and 12th months of the year.

Evolution of the Gregorian Calendar

The early Egyptian calendar started as a simple calendar of 12 months of 30 days each, so a year only had 360 days. This was 5.2422 days shorter than a solar year, which caused holidays to rotate through all of the seasons over a period of 69 years, 360/(365.2422 - 360).

About the year 4000 B.C. (before the common era or B.C.E.) the Egyptian calendar was modified, and 5 days were added at the end of each year. This was called the vague year, and it fell behind the solar year by one day every four years. This system was still not perfect, and a holiday with a fixed date still rotated through all the seasons, except, instead of taking 72 years, it took almost 15 centuries. Since a holiday moved back about one day every 4 years, it came back to its original position after 1,507 years, 365/(365.2422 - 365).

This 1,507 year period was known to the Egyptians, who gave it a name—the Sothic period. This came from the Egyptian name Sothis for the star Sirius because the Egyptians knew that the flooding of the Nile began each year at the time of the heliacal rising of Sirius. The Egyptian calendar was divided into 3 seasons of 4 months each. Every 1,507 years the heliacal rising of Sirius fell on New Years day and marked the beginning of a new Sothic period.

In 238 B.C. the Greeks, who had also based their calendar on the Egyptian’s, were the first to correct the vague year by adding an extra day every four years. The Egyptians refused to accept this reform, and when a third Sothic period ended in 139 A.D., they continued to use the vague year. The Romans did not adopt the reform until 46 B.C. when Julius Caesar reformed the Roman calendar.

The Julian Reform

When Julius Caesar reformed the calendar in 46 B.C. (1 January A.U.C.—Ab urbe condita, from the traditional founding of the city of Rome), it was decided that the vernal equinox would fall on March 25th. To do this, 85 days were added to the year 46 B.C. The civil year was then fixed at 365 days with an additional day intercalated every 4 years after February 24th, leap year. In addition, January 1st became the first day of the year, thus creating a discrepancy in the names of the months, which were then out of order. These changes were known as the Julian reform.

The Julian year of 365.25 days was still a little longer than the solar year of 365.2422 days. This difference accrued an error of about 3 days every 4 centuries, and a holiday would then rotate through the year in about 46,827 years, 365.25/(365.25 - 365.2422).

The Gregorian Reform

By the time of Pope Gregory XIII, in the late 16th century, the deviation had accrued to about 10 days; the equinox of 1582 fell on March 11th. The true length of the solar year was widely known by this time, and Pope Gregory instituted a reform to the calendar. He cut 10 days from the year 1582, so that in the next year, the equinox would fall on March 21st. The day after Thursday, 4 October 1582, became Friday, 15 October 1582, the first day of the Gregorian calendar. Century years would only be a leap year if divisible by 400 so that 1900 was not a leap year, but 2000 will be, thereby giving the system a 400 year cycle of 146,097 days (400 * 365 + 97) and an average year of 146,097/400, or 365.2425 days. This reform was known as the Gregorian reform, and this is the calendar we use today.

There is still a remaining inaccuracy in that the Gregorian year is equal to an average of 365.2425 days and is, therefore, longer than the solar year by .0003 days per year. The excess amounts to about 3 days every 10,000 years, and a holiday will now take about 1.2 million years to rotate through the year, 365.2425/(365.2425 - 365.2422). This inaccuracy is not considered important because, in reality, the actual length of the solar year varies and the rules to predict the variance are not precisely known.

However, the Gregorian reform was not accepted uniformly. France and the Netherlands adopted the reform in December, 1582; the Catholic states of Germany in 1584; Poland in 1586; the Protestant states of Germany and Switzerland in 1700; Sweden, England and its colonies, including America, did not adopt the reform until 1752. Russia waited until 1918 and Bulgaria until 1920. The calendar system being cited is important information if you are working with dates that extend back into these transitional years. Notice, again, that 1582 is also the beginning of the valid date range in the SAS system.
Future Calendar Reforms

There has been talk about reforming the current Gregorian calendar. Despite its accuracy there are still a few defects. The number of days in a month varies by about 12%, which must be considered when producing any statistics on a monthly basis. In addition, the day of the week for each date changes from year to year. Further, finding the day of the week on which a date falls without a calendar at hand is difficult. Many reforms have been proposed to remedy these defects. The Pope, nevertheless, has vowed to remain uninvolved as long as Easter does not again wander into winter.

Partial Dates

Now that we have a thorough understanding of the concept and the history of dates, we can return to the main question--how does the SAS system represent partial dates? The answer is that SAS does not have a method of representing partial dates, but we can develop one.

<table>
<thead>
<tr>
<th>Precision Code</th>
<th>Format</th>
<th>Description</th>
<th>Quality</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>YYMMDD</td>
<td>Complete date</td>
<td>Valid</td>
</tr>
<tr>
<td>1</td>
<td>YYMM</td>
<td>Year and Month</td>
<td>Valid</td>
</tr>
<tr>
<td>2</td>
<td>YYDD</td>
<td>Year and Day</td>
<td>Valid year only</td>
</tr>
<tr>
<td>3</td>
<td>YY</td>
<td>Year only</td>
<td>Valid</td>
</tr>
<tr>
<td>4</td>
<td>MMDD</td>
<td>Month and Day</td>
<td>Not Valid</td>
</tr>
<tr>
<td>5</td>
<td>MM</td>
<td>Month only</td>
<td>Not Valid</td>
</tr>
<tr>
<td>6</td>
<td>DD</td>
<td>Day only</td>
<td>Not Valid</td>
</tr>
</tbody>
</table>

Invalid dates are one thing. At least, you can claim that they are categorically wrong and, therefore, can not be reported. The data must then either be corrected or ignored. Partial dates are another story--they come up in real life, and they must be reportable without much heartache.

Defining Partial Dates

Defining the difference between a partial date and an invalid date is necessary. If a complete date can be placed in the YYMMDD format, then only six possible partial date formats exist, as illustrated in table 1.

Only the partial dates that can be described by a continuous, finite range will be considered valid. The YY and YYMM formats will, therefore, be the only types of valid partial dates. The formats MM, DD, and MMDD will not be considered valid because they can occur in any year and, in theory, represent an infinite number of possible valid dates. The format YYDD is not considered valid because it could represent up to 12 specific dates in any one year and is not a continuous range although it could be truncated to conform to the YY format.

An argument could possibly be made for the MM and MMDD formats because they still represent a minor amount of information, such as the season of the year. However, trying to make use of these formats would complicate any methodology and be of limited use in analysis.
A Systematic Approach to Partial Dates

I would like to present a systematic approach for handling partial dates from within the framework of SAS date values. This approach involves forcing valid partial dates to be complete and invalid partial dates to missing so that any date, partial or full, can be stored and manipulated as a SAS date value. This necessitates associating a precision field with every date value field that could contain a partial date. For example, if DOB was a date variable in a database, then perhaps DOB_P, would be the variable indicating the precision of DOB, as illustrated in table 2.

When forcing partial dates to be complete, a documented methodology should be used. Depending on the context, setting the missing values to be as early or as late as possible in order to express the most conservative or optimistic estimate may be desirable. In some instances a median value may be used to give the best approximation.

For example, '89JAN', could be forced to '89JAN31', '89JAN01', or '89JAN15', to represent the latest, earliest,

<table>
<thead>
<tr>
<th>TABLE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample Input</td>
</tr>
<tr>
<td>Unedited Dates</td>
</tr>
<tr>
<td>----------------</td>
</tr>
<tr>
<td>DOB_TEXT</td>
</tr>
<tr>
<td>17MAY1964</td>
</tr>
<tr>
<td>JUN1928</td>
</tr>
<tr>
<td>12 1987</td>
</tr>
<tr>
<td>1961</td>
</tr>
<tr>
<td>01APR</td>
</tr>
<tr>
<td>JAN</td>
</tr>
<tr>
<td>15</td>
</tr>
</tbody>
</table>

Because the SAS system has a vast selection of useful date functions available, it is worthwhile to attempt to represent partial dates from within the framework of SAS date values. Specific algorithms could be developed to perform the same functions as provided by the SAS system while allowing for partial dates, but this would involve a great amount of programming effort to accommodate a small percentage of data. Remember that the SAS system already contains all of the groundwork for manipulating dates. Any effort to write new date functions would violate the primary law of procedural abstraction: avoid duplication of effort.

### TABLE 3

#### Rules for Completing Partial Dates

<table>
<thead>
<tr>
<th>Precision Code</th>
<th>Format</th>
<th>Data Class</th>
<th>Rules: Used only when missing data cannot be obtained</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>YYMMDD</td>
<td>ALL</td>
<td>Use complete date</td>
</tr>
<tr>
<td>1</td>
<td>YYMM</td>
<td>1</td>
<td>Use median day of month, 15</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Set to first day of month</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Set to last day of month</td>
</tr>
<tr>
<td>2</td>
<td>YYDD</td>
<td>1</td>
<td>Discard days and follow rules for YY format</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>YY</td>
<td>1</td>
<td>Use median month and day of year, 01 July</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>Set to first month and day, 01 January</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>Set to last month and day, 31 December</td>
</tr>
<tr>
<td>4</td>
<td>MMDD</td>
<td>ALL</td>
<td>Set to missing</td>
</tr>
<tr>
<td>5</td>
<td>MM</td>
<td>ALL</td>
<td>Set to missing</td>
</tr>
<tr>
<td>6</td>
<td>DD</td>
<td>ALL</td>
<td>Set to missing</td>
</tr>
</tbody>
</table>

59
or median estimated dates. For dates containing only the year and the month, the median day could be computed for each month, or an average median of 15 could be used. For simplicity, in this paper 15 will be considered the median day of every month. In the event that only the year is provided, the median month and day would be 'JUL01', on both a standard and a leap year. This corresponds to the mid-point of a year, day 182.5 (365/2) or day 183 on a leap year.

These rules, as summarized in Table 3, must be associated with each date field that could contain partial dates. In other words, each date field in the database must be classified as either containing complete dates only or containing both complete and partial dates. Those fields, which could contain partial dates, must be further classified as to how the partial dates were completed--first, last, or median. These decisions should be made with input from the appropriate users of the data. The rules could then be applied programmatically to missing dates by the appropriate data management personnel. If the rules were applied consistently within an organization and well documented, they would provide a strong foundation for an organization wide, systematic approach for reporting partial dates. A macro for completing partial dates is included as appendix 1.

The precision field associated with each date consists of a one byte code indicating the format of the date, where '0' indicates a complete date and the codes for partial dates range from '1' to '6', a null field indicating a missing date. Programs can then be modified to either exclude or include estimated dates from any analysis, and reports could flag partial dates as estimated. In addition, the actual partial date can be included in listings by converting the completed SAS date value into a character string according to the format indicated by the precision code. Appendix 2 contains a macro used to convert completed partial dates back into an incomplete format for printing purposes.

**Conclusion**

Adopting a systematic approach for reporting partial dates will have benefits in many areas, and industries. Programmers will benefit by being freed from the drudgery of repeatedly writing adhoc code to handle individual cases of partial dates. The organization will benefit by being able to make use of information that was previously unusable, which, in turn, will improve the quality of output. Customers will benefit by being aware of an organization standard which guarantees consistency across projects. Finally, the ability to use the SAS system's existing framework for manipulating dates will allow a comprehensive set of existing tools to be used as is, saving vast amounts of programming effort that may have been needed if another approach had been taken.

Like our ancestors, who dared to make radical changes to their calendars so that we would not be forced to hunt through the snow for Easter eggs, we have the opportunity to make a fundamental change in the way our organizations handle dates. This continuing evolution will ensure that our descendants will have a calendar in which they can depend, for Easers in spring and a consistent method for reporting partial dates.

**References**


Grolier Inc. (1990), *Encyclopedia Americana*, Volume 5, 185-191, Danbury, CT: Grolier Inc.


**Acknowledgments**

SAS and SAS/ACCESS are registered trademarks or trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

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

Thanks to Wendy Blumberg, Margie DeStefano, Allan Glaser, Jill Golde, Anthony Palumbo and Nate Williams.

**Contact**

Jerry Kagan
SmithKline Beecham Pharmaceuticals
Mail Code #FF0605
P.O. Box 1510
King of Prussia, PA 19406-0939
(215) 832-3628

Internet Electronic Mail: kaganj1%solar.DNET@SMITHKLINE.COM
Appendix 1 - Reading Partial Dates

%*************************************************************%
% Library:  my_Code
% File Name: Dateinp.SAS
% Language: SAS 6.07
% Purpose:  This SAS macro is called from within a data step
% and will read character dates of the form YYMMDD
% and convert them to SAS date values according to
% the rules for completing partial dates. A variable
% will be set to indicate the precision code for the date.
% The rules used will vary according to the data
% classification of the date, which can be a 1, 2, or 3
% indicating the median, minimum, or maximum
% completed date, respectively. The class will default
to 1, the median, if not specified.
% Protocol:
% Macro Parameter Definitions:
% Macro Parameter Definitions:
% charDate - Variable containing character date input in
% YYMMDD format
% sasDate - SAS date variable
% precise - One byte field for the precision of the SAS date
% class - 1, 2, or 3 indicating the median, minimum, or
% maximum approximation is to be used.
% %macro Dateinp (charDate, sasDate, precise, class);
% %** Check for any errors in parameter list before processing begins
% %** Initialize error flag:
% %if %length(%quote(&charDate)) = 0 %then
% %do;
% %let _error = 1;
% %end;
% %if %length(%quote(&charDate)) = 0 %then
% %do;
% %put ERROR: Missing charDate in macro Dateinp;
% %let _error = 1;
% %end;
% %if %length(%quote(&sasDate)) = 0 %then
% %do;
% %put ERROR: Missing sasDate in macro Dateinp;
% %let _error = 1;
% %end;
% %if %length(%quote(&precise)) = 0 %then
% %do;
% %put ERROR: Missing precision in macro Dateinp;
% %let _error = 1;
% %end;
% % Three classes were defined:
% %* 1 - median date, 2 - earliest date, 3 - latest date;
% %if &class = %then
% %let class = 1;
% %else if &class > 3 | &class < 1 %then
% %do;
% %put ERROR: Class out of range in macro Dateinp;
% %let _error = 1;
% %end;
% % if no parameter errors, then begin processing.
% %if _error = 0 %then
% %do;
% %** Split up character date into numeric components;
% length _Year _Month _Day 2;
% _Year = input(substr(&charDate,1,2),2.);
% _Month = input(substr(&charDate,3,2),2.);
% _Day = input(substr(&charDate,5,2),2.);
% label &sasDate = "SAS date value, completed from &charDate;"
% %end;
% %** YYMMD - Full date, use as is, set precision code to 0.
% if _Year ^= . & _Month ^= . & _Day ^= . then
% do;
% &precise = '0';
% &sasDate = mdy(_Month, _Day, _Year);
% end;
%*************************************************************%
% Example(s): %Dateinp( dob_char, dob_sas, dob_P );
% %Dateinp( dob_char, dob_sas, dob_P );
% %Dateinp( dob_char, dob_sas, dob_P );
% %Author: Jerry Kagan
% %Date: 1 March 1993
% % Reviewed by:
% %Revisions:
% %
%* YYMM - Year & month only, use median, minimum, or maximum based on class. Use 15 as the median day of any month. Set precision code to 1.

else if _Year='.' & _Month='.' & _Day='.' then
  &_precise='1';
  select (&class);
  when (1) &sasOate = mdy(_Month, 15, _Year); %' Med;
  when (2) &sasOate = mdy(_Month, 1, _Year); %' Min;
  when (3) &sasOate = intrx('month', mdy(_Month, 1, _Year), 1)-1; %' Max;
  &sasOate = ;
end;

%* YY_DD - Year & day only, discard days and follow the rules for the year only format. Set precision code to 2.

else if _Year='.' & _Month='.' & _Day='.' then
  &_precise='2';
  select (&class);
  when (1) &sasOate = mdy(7, 1, _Year); %' Med, Jul01;
  when (2) &sasOate = mdy(1, 1, _Year); %' Min, Jan01;
  when (3) &sasOate = mdy(12, 31, _Year); %' Max, Dec31;
  &sasOate = ;
end;

%* YY__ - Year only, complete date using median, minimum, or maximum date based on class. Set precision code to 3.

else if _Year='.' & _Month='.' & _Day='.' then
  &_precise='3';
  select (&class);
  when (1) &sasOate = mdy(7, 1, _Year); %' Med, Jul01;
  when (2) &sasOate = mdy(1, 1, _Year); %' Min, Jan01;
  when (3) &sasOate = mdy(12, 31, _Year); %' Max, Dec31;
  &sasOate = ;
end;

%* MMDD - Missing year, set code to 4 and SAS date to missing.

else if _Year='.' & _Month='.' & _Day='.' then
  &_precise='4';
  &sasOate = ;
end;

%* MM - Months only, set precision to 5 and date to missing.

else if _Year='.' & _Month='.' & _Day='.' then
  &_precise='5';
  &sasOate = ;
end;

%* DD - Days only, set precision to 6 and date to missing.

else if _Year='.' & _Month='.' & _Day='.' then
  &_precise='6';
  &sasOate = ;
end;

%* DO - Days only, set precision to 6 and date to missing.

else if _Year='.' & _Month='.' & _Day='.' then
  &_precise='6';
  &sasOate = ;
end;

%* Drop all temporary variables.

%end;
%mend Datelnp;
Appendix 2 - Writing Partial Dates

%Library: my_Code
%File Name: DateOut.SAS
%Language: SAS 6.07 Macro Language
%Purpose: This SAS macro is called from within a data step and will convert completed partial dates using an associated precision code into a character variable suitable for printing. Missing days', months', or years' positions are left blank. Dates can be converted into the various formats but will default to date7. Valid formats are: date7., dates9., mmddyy6., mmdyy6., yymmd6., yymmd6.
%Protocol: %DateOut ( SAS_Date , Precision_Code , Character_Date , [ DATE7. , DATE9. , MMDDYY6. , MMDDYY6. , YYYMMD6. , YYMMMD6. ] );
%Example(s): %DateOut( dob, dob_P, dobPrt. yymmdd. );
%Author: Jerry Kagan
%Date: 2 March 1993
%Reviewed by:
%Revisions:
%Parameter Definitions:
% sasDate - SAS date variable.
% precise - One byte field containing the precision of the SAS date.
% charDate - A character date for printing.
% format - A SAS date format from the list above.
%Macro Parameter Definitions:
% Macro Parameter Definitions:
% sasDate - SAS date variable.
% precise - One byte field containing the precision of the SAS date.
% charDate - A character date for printing.
% format - A SAS date format from the list above.
%macro DateOut ( sasDate, precise, charDate, format ) ;

%* Check for any errors in parameter list before processing begins:
%* Initialize error flag:
%* If length(&sasDate) = 0 then:
%* Put ERROR: Missing sasDate in macro DateOut;
%* End;
select (&precise);
when ('0') do:
  _Year4 = put( year( &sasDate ), 4 );
  _Year2 = substr( _Year4, 3, 2 );
  _nMonth = put( month( &sasDate ), 2 );
  _Day = put( day( &sasDate ), 2 );
end;
when ('1') do:
  _Year4 = put( year( &sasDate ), 4 );
  _Year2 = substr( _Year4, 3, 2 );
  _nMonth = put( month( &sasDate ), 2 );
  _Day = put( day( &sasDate ), 2 );
end;
when ('2', '3') do:
  _Year4 = put( year( &sasDate ), 4 );
  _Year2 = substr( _Year4, 3, 2 );
end;
when ('4', '5', '6') do:
  _nMonth = put( month( &sasDate, monname3 ), 2 );
end;
otherwise do:
  %put Warning: &precise out of range in macro DateOut,
  &charDate set to ""; %...
end;
end:

&charDate = _nMonth || _Sepor || _Day || _Sepor || _Year2;
%end:

%else if &format = YMDMD6. %then %do;
  &charDate = _Year2 || _nMonth || _Day;
%end;

%else if &format = YMDMD8. %then %do;
  &charDate = _Year2 || _Sepor || _nMonth || _Sepor || _Day;
%end;

%end;

%* Assemble &charDate according to format.

%* ....
* Label &charDate = 'Character date value, converted from' &sasDate;

%* ....

&&format = DATE7. %then %do;
  &charDate = _Day || _nMonth || _Year2;
%end;

%else if &format = DATE9. %then %do;
  &charDate = _Day || _nMonth || _Year4;
%end;

%else if &format = MMDDYY6. %then %do;
  &charDate = _nMonth || _Day || _Year2;
%end;

%else if &format = MMDDYY8. %then %do;

%end;

%end;